Physics of a stern-fixed single-blade sculling oar like a yuloh

Most discussions about a yuloh jump right into blade profile, blade-to-loom angle, lanyard fixing point and the like. Those details can obscure some of the first principles which ought to be considered, well, first.

Propelling a boat forwards is all about accelerating water backwards. Propelling a boat forwards efficiently is all about matching the power produced by the engine (a human) to the power absorbed by the water.

Let's consider a traditional pair of side oars. This is a brute force way of accelerating water forwards. The rower pulls the inboard ends of the oars towards himself; the blades on the other ends dig into the water and push it backwards. Not all of the effort goes into pushing water backwards. Since the blades travel in circular arcs when seen from above, water is being pushed outwards from the direction of travel as well as backwards at the start of a stroke. At the end of the stroke, water is being pushed towards the centerline as well as backwards. In fact, the direction in which the water is being pushed is ideal only at the moment when the oars extend perpendicularly out from the centerline. The effort expended pushing water outwards and inwards is wasted insofar as the goal of propulsion is concerned. Experience tells us this waste is acceptable because the propulsion system as a whole is pretty effective.

Furthermore, the oars produce benefit during only one-half of each stroke. On the return stroke, the rower has to lift the blades clear of the water and swing them back to their starting positions. All of the effort and time needed for the return half of the stroke is a waste from the point-of-view of the propulsion system. But even that waste is acceptable.

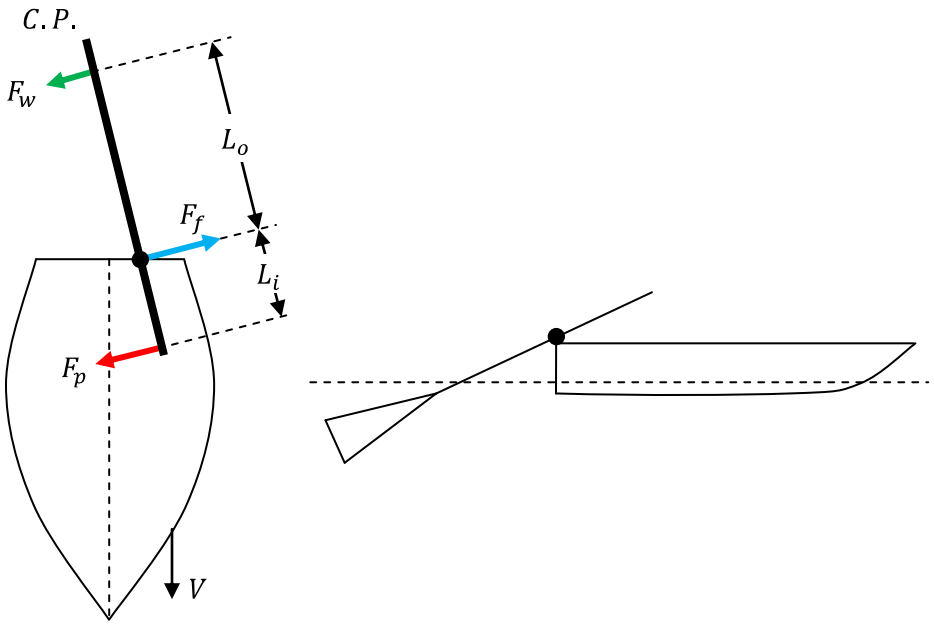
The length of the oars and the relative position of their fulcrums have to be chosen carefully. The length of the arc through which the inboard ends travel has to be appropriate for a rower. An inboard swing of only a foot would be useless. It would waste the potential we expect a normal-sized rower can deliver if he is allowed to pull through three or four feet. Now, consider the other end of the oar. It would be fantastic if the blades were the size of a 4 foot by 8 foot sheet of plywood. Huge amounts of water could be moved. But intuition tells us that would be ineffective. Moving such large volumes of water takes a lot of energy. It might take an Olympic rower five seconds to pull such oars through one stroke. We all know that a more comfortable and sustainable rate for humans is a stroke every two seconds or so. The length of the outboard ends of the oars has similar effects. Pulling a stroke on 50 foot oars would be hard, to say nothing of the extra weight of oar which would need to be hauled.

As children, most of us saw innumerable images of side oars and their use, so we have a feeling for what oars look like and what they do. Yulohs, not so much. Take oar length as an example. We can all reason out, as I have just done, the factors which cause side oars to have the lengths they do. Yulohs are no different. Certain factors affect their lengths; we just have to figure out what those factors are. Yulohs are no more arbitrary in their design than side oars.

(In the following, I am going to use the words "sculling oar", "scull" and "yuloh" interchangeably.)

Analysis of a rudder-type yuloh

I am going to start by looking at the yuloh as a type of rudder, whose blade is a thin sheet of aluminum, say, which remains at all times vertical with respect to the water surface. It is waved from side-to-side like one would oscillate a rudder from side-to-side by pulling the tiller back and forth. The only meaningful difference is that the center of pressure on the blade is a further distance aft of the transom than a typical rudder. The following figure shows the situation from above and from the side.



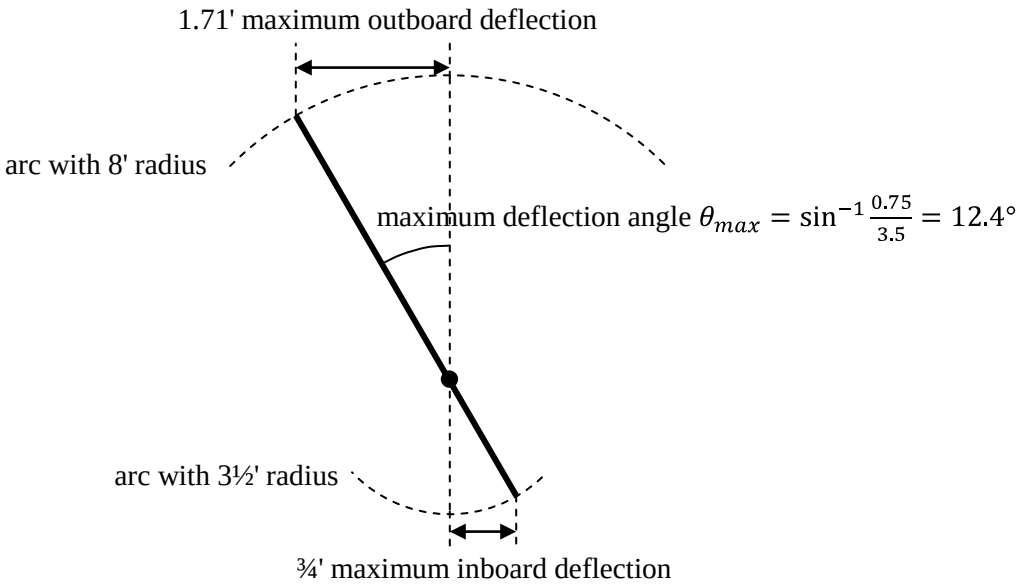
I have put the fulcrum a little bit to port of the boat's centerline. The outboard and inboard lengths of the yuloh, L_o and L_i, respectively, are not their end-to-end lengths but rather the distances from the fulcrum to the center of pressure of the blade ($C.P.$) on the one hand and to the person's center-of-effort on the other. The water exerts a net force F_w on the blade when the sculler pulls with force F_p on the loom. The subscript p of the latter could stand for person, or push/pull or propulsion. The force which counteracts the other two is the force of the fulcrum on the scull F_f. All three of these forces are forces which act on the scull. Each of them has a reaction force, which is approximately equal in magnitude.

It is the reaction to force F_w which pushes the water towards the rear (and to the side). It is the reaction to force F_f which pushes the boat forwards (and to the side as well). And, it is the reaction to force F_p which pulls the person towards the port side and requires that he take a stance in preparation for his pull.

The directions of the forces in the figure are not very useful. It looks like the blade does a better job pushing water towards the right (in the figure) than downstream. The force at the fulcrum looks much more likely to yaw the boat than to push it forwards. This is exactly what happens when the scull is swept while the is in irons. A good strong pull or push on the tiller will not do much more than move the stern from one side to the other. It is possible to eke out some forward momentum by operating the tiller with care: slowly moving it off-center and then yanking it straight. But that is not very effective. It is like trying to use the rudder as a paddle.

Let's see if this system works better when the boat already has some forward speed. I will try to estimate some realistic speeds. Assume the boat is moving forwards at one and one-half knots, which is equivalent to 2.53 feet per second. To quantify the angular motion, assume the boat is a 16-footer and uses a yuloh with a total length of 14 feet, of which 4 feet are inboard from the fulcrum and 10 feet are outboard. (I will refer to the inboard end of the yuloh as the "loom".) From these dimensions, I have estimated that the center of pressure is 8 feet from the transom (L_o) and that the inboard length is 3½ feet (L_i). Lastly, I have assumed that the sculler is working 30 cycles per minute through strokes with a length of 1½ feet.

The geometry of the strokes is shown in the following figure. When the scull is at the end of a stroke, the inboard end is ¾ of a foot from the centerline and the center-of-pressure is 1.71 feet from the centerline.



The angle which the scull makes with respect to the centerline at the end of a stroke is 12.4°. The blade's center of pressure does not travel in a straight line, of course, but in a circular arc. The length of the circumference of an arc having an eight foot radius, from 12.4° on one side of the centerline to 12.4° on the other, is given by:

$$\text{arc length} = \frac{24.8°}{360°} \times 2\pi \times 8 = 3.46 \text{ feet}$$

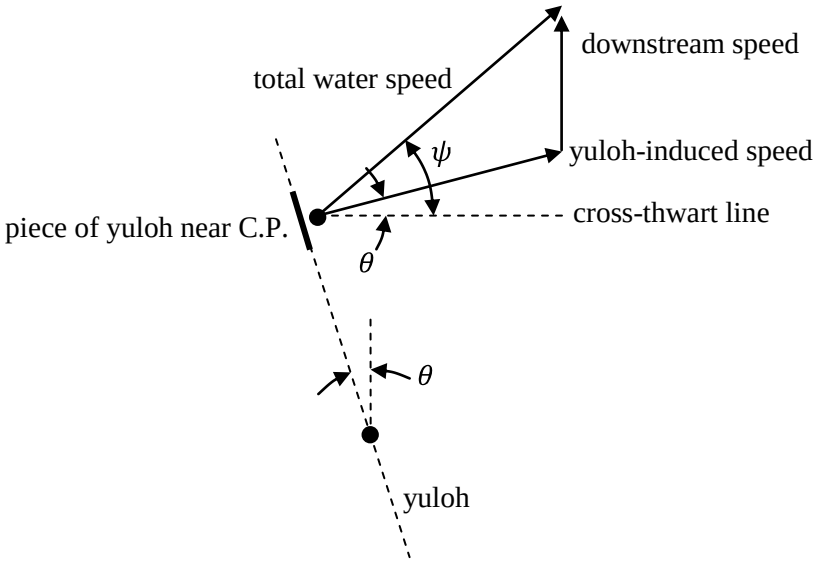
This is the distance through which the center of pressure travels during one-half of a stroke. Since there are 30 strokes per second, the duration of a half-stroke is one second. The tangential speed of the center of pressure during the stroke is therefore equal to:

$$\text{speed of C. P.} = \frac{3.46 \text{ feet}}{1 \text{ sec}} = 3.46 \text{ feet per second}$$

This is only a first approximation. It assumes that the person pulls his end of the scull through a circular arc at a constant speed, with no diminishment at the turning points. Furthermore, this estimate applies only to points on the blade which are eight feet from the fulcrum. Points further out on the bade will travel faster; points closer in will move more slowly.

The significant conclusion is that the arc-speed of the center of pressure (3.46 fps) is of the same order of magnitude as the forward speed of the boat (2.53 fps).

The following figure shows how we can use vector addition to combine the speed of the blade with the speed of the boat. It is easiest to think of the boat being held at rest, by its painter, in a stream flowing by at 1½ knots. The dynamics of the boat in the water are the same whether the water is calm and the boat plows through it, or whether the boat is held still and the water flows under it. It is reasonable to imagine that bits of water which are in close proximity to the pressure-side of the blade will be accelerated to the same speed as the approaching blade. This extra speed will be added to the speed the bits already had in the downstream direction.



When the yuloh is at some angle θ with respect to the boat's centerline, a piece of the yulah near the center of pressure will induce a speed which lies at an angle θ aft of the perpendicular to the centerline. I have called this line, which is perpendicular to the centerline, the "cross-thwart line".

But, when the existing downstream speed of the water is added, the total speed of the water near the center of pressure will be at angle ψ, which is greater than θ, aft of the cross-thwart line.

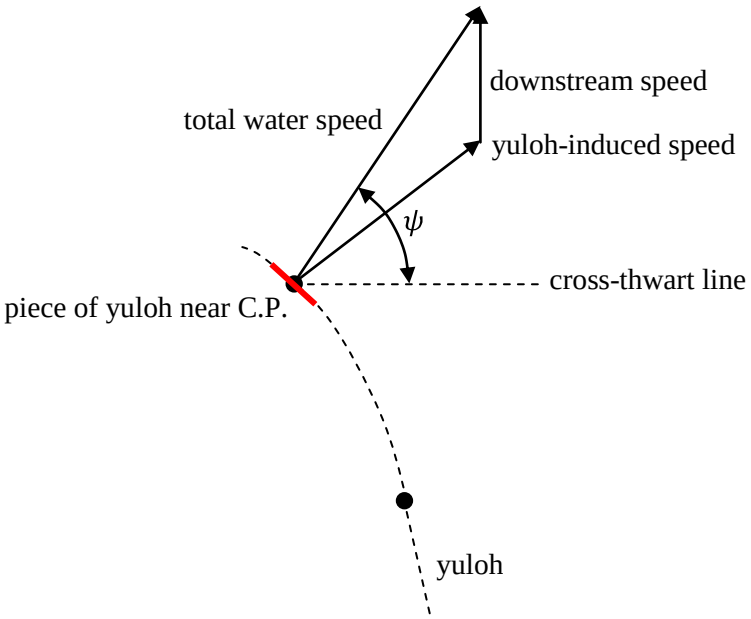
We have to go through two thought processes to apply this result.

1. Firstly, we need to recall that the physics are the same however the forward speed of the boat arises. However it arises, the angle at which water peels off the yuloh will be at angle ψ, which is greater than θ. In other words, the total angle in which the water is sent is further aft than when the boat is at rest.

2. It is very difficult to work through the mechanics by which the water exerts pressure on the blade. But it very easy to figure out the net result, if we step back for one moment and look at a larger universe. Consider a big spot on a calm lake before the boat passes. The water is at rest; it does not have any kinetic energy. Now consider the same calm spot after the boat passes through. During the passage, the water received energy. Bits of water were forced to move in a direction which was angled ψ aft of the line of the boat's passage. (Water is a viscous fluid, so that the kinetic energy which was added by the boat was subsequently dissipated as heat, and the water returned to its pre-passage calm.) But, the boat added energy, and momentum, to the water. Momentum was conserved during the passage. Whatever momentum the boat added to the water was added to the boat's momentum in the other direction. The more backwards momentum the boat added to the water; the more forward momentum the water added to the boat. (It takes energy to overcome the water's drag on the boat, so momentum which is added to the boat gets used up overcoming drag. New momentum needs to be added continuously to keep the boat moving.)

The bottom line is this: the bigger angle ψ can be made, the more forward momentum is added to the boat. It follows that a rudder-like scull will become more effective as the boat's speed increases.

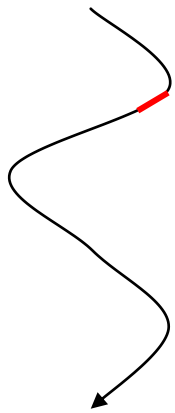
There is a second way to make a rudder-like scull more effective: by making it flexible. Consider the tip of the flexible yuloh shown in the following figure.



Because the tip bends, water near the tip is directed even more closely downstream than before. With a flexible enough blade, one could pull the inboard end (the "loom") past the centerline of the boat while the tip is still deflecting water downstream.

There is an offsetting disadvantage. Consider the scull at the start of a stroke, when the angle the scull makes with the centerline of the boat is at its maximum. When the sculler begins the stroke, the tip does not move. During the first phase of the stroke, the energy which the sculler is putting into the system is being stored as potential energy in the shaft of the yuloh, as it bends. Only after the internal stresses in the bending shaft have increased to a certain point will the sculler's power begin to find its way into moving the tip. The shaft will remain bent until the sculler releases his pull near the end of the stroke. At this time, the potential energy in the shaft is going to be released, and cause the tip to keep moving. Unfortunately, the potential energy is going to be wasted or, even worse, release itself at a detrimental angle. If the sculler has pulled too far through the centerline, the tip will flick in the wrong direction, driving water upstream.

However, it should be noted that many fish swim using exactly this mechanism. They have a natural ability to match the side-by-side speed of their tail fin to their forward speed in such a way that the completion of a "stroke" is not an uncontrolled upstream flick. They have muscles along their body which let them control the shapes of their rear ends in a way that is not possible for a fixed construction yuloh capable only of rotation around the fulcrum.



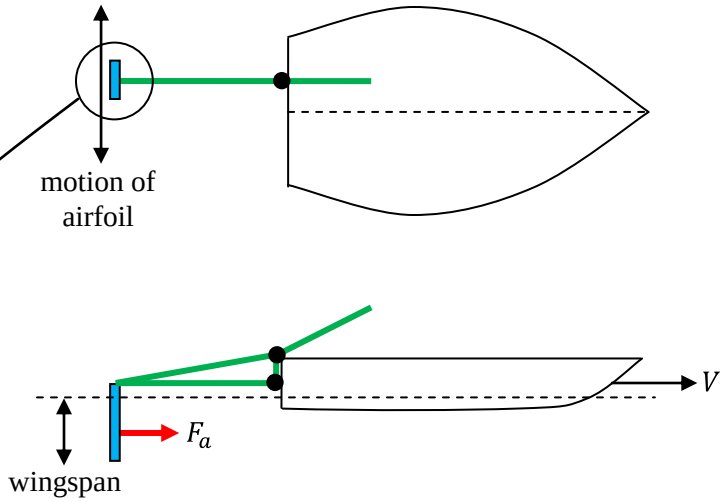
We could try to learn from the fish, and design a yuloh in which each section, like the red section shown in the figure, travels through the water in a sinusoidal pattern. The stiffness of the blade would have to change continuously along its length. Even then, the resulting scull would only be optimal for a certain combination of forward speed, amplitude of side-to-side oscillation and stroke frequency.

Perhaps such a yuloh could be made. Or, perhaps not. The fundamental problem is pretty simple. An oar with its long axis parallel to the boat's centerline is just the wrong place to start if the objective is to push water in the direction of the long axis.

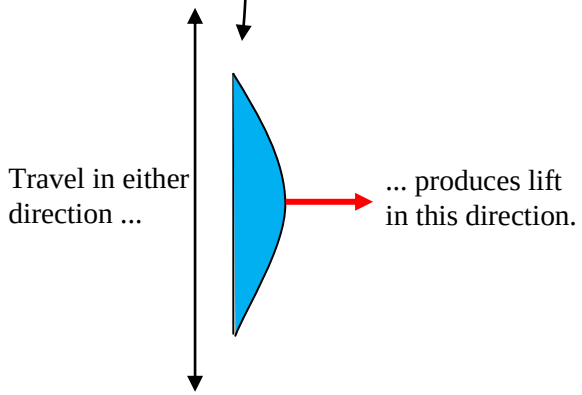
Analysis of an airfoil-type yuloh

Let's look at something completely different. Airplanes have a long and proven history of producing a force in one direction (upwards, and called "lift") which is the result of motion in the perpendicular direction (which, for an airplane, is the direction of flight). A yuloh based on the principles of flight is shown in the following figure.

The airfoil is the blue box. It is a small wing whose long axis is vertical and points straight down into the water. The wingspan is identified in the lower part of the figure as the length of the wing which is submerged. If any part of the wing is not in the water, its motion will not add anything to the propulsion. In use, the airfoil moves back and forth across the width of the boat, as shown by the double-sided arrow in the top part of the figure. Movement of the airfoil will produce lift (hopefully) which points straight in the direction of the boat's travel. The lift force is identified as the red arrow F_a in the lower part. The rest of the yuloh is comprised of four straight sticks, or bars, which are shown in green. They are simply structural elements whose purpose at this point is to transfer the forces exerted by the person into side-to-side motion of the airfoil. Think of the black dots as pintles, which hold the airfoil in its vertical position while allowing it to move sideways.



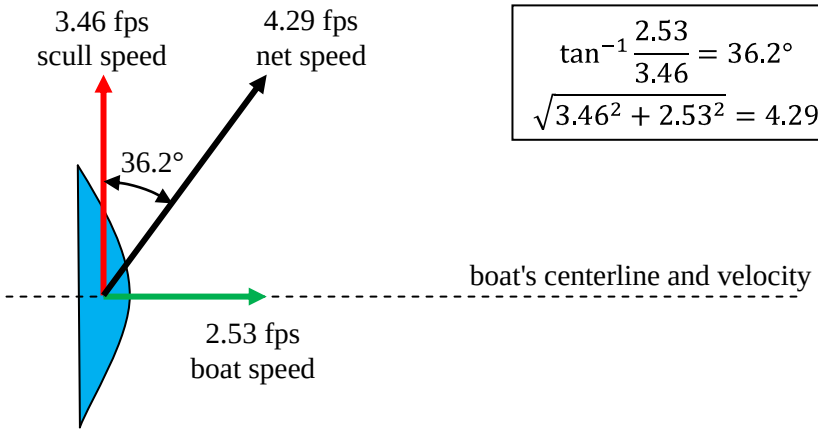
We want the airfoil to have a symmetrical profile so that it will produce lift when it moves towards either side. It should look something like the following. It is important that the curved side be facing the direction of travel. It is important that the flat side, or flatter side, face aft. It is the differential curvature between the front and back sides which causes the airfoil to produce lift. Furthermore, the profile needs to be sharp at both edges. The Kutta condition, on which a lot of aerodynamics rests, shows that a sharp trailing edge is a necessary condition for initiating the vortex which leads to circulation around the profile. A rounded trailing edge leads to a poorly-defined initial vortex and reduced or unstable lift. On the other hand, a rounded leading edge is better than a sharp one, because it results in less drag. Since we want our yuloh to be equally powerful and productive in both directions of a stroke, we need both edges to be the same. Since it is more important to have a sharp trailing edge than a rounded leading edge, we are going to have to make both edges sharp.



We want to make the most effective airfoil we can. In airplane-speak, we want the highest coefficient of lift and the highest lift-to-drag ratio we can get for the given flight conditions. Since the fluid the airfoil is flying through is water, some of the things we know about airplane wings may have to change. For example, the airfoil is going to be flying at extremely low speeds compared with air travel. In the

analysis above, we encountered speeds like two and three feet per second, or between one and two miles per hour.

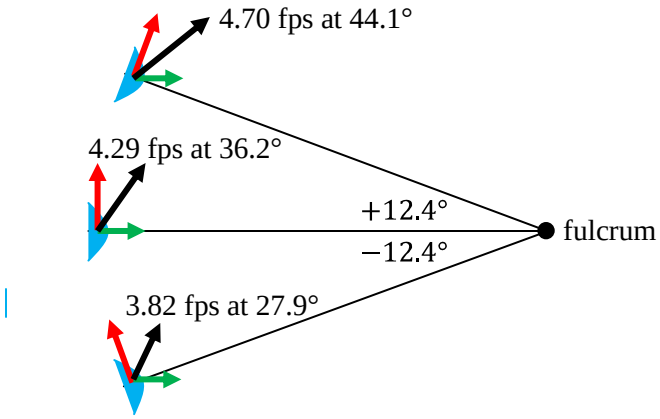
The figures above suggest a striking difference from typical airplane flight: that the airfoil will be flying at negative angles of attack. Let's look into this. Let's assume once again that the boat is moving forward at 1½ knots, or 2.53 feet per second. Let's keep the same stroke parameters we looked at before, namely, that the center of pressure (which is now located at the center of the airfoil) travels through its circular arc at 3.46 feet per second. (In due course, we are probably going to have to adjust the dimensions of the yuloh, but let's not get too far ahead of ourselves yet.) We can use these two speeds to get an idea of the direction from which water approaches the airfoil.



$$\tan^{-1}\frac{2.53}{3.46} = 36.2°$$
$$\sqrt{3.46^2 + 2.53^2} = 4.29$$

Water is approaching the airfoil from an angle 36.2° above the line of its flat surface. This is well and truly outside the range of angles of attack encountered in traditional airplane flight.

This diagram applies only when the scull is aligned with the centerline of the boat. The angle of attack will change as the yuloh sweeps through a half-stroke. The extreme angles of attack will occur when the yuloh is at its maximum deflections from the centerline. To look at the extrema, let's once again take over a result from the analysis of a rudder-type scull. There, we found that the extreme angular deflections of the yuloh were $\pm 12.4°$. The following figure shows pictorially the vector addition of the boat speed (the green arrow) and the airfoil's forward speed (the red arrow) at the extremes of a scull stroke. The black arrow in each case is the relative velocity of the water relative to the airfoil's flat surface. One can see that the relative speed varies from 3.82 feet per second to 4.70 feet per second and the angle of attack varies from minus 27.9° to minus 44.1°.



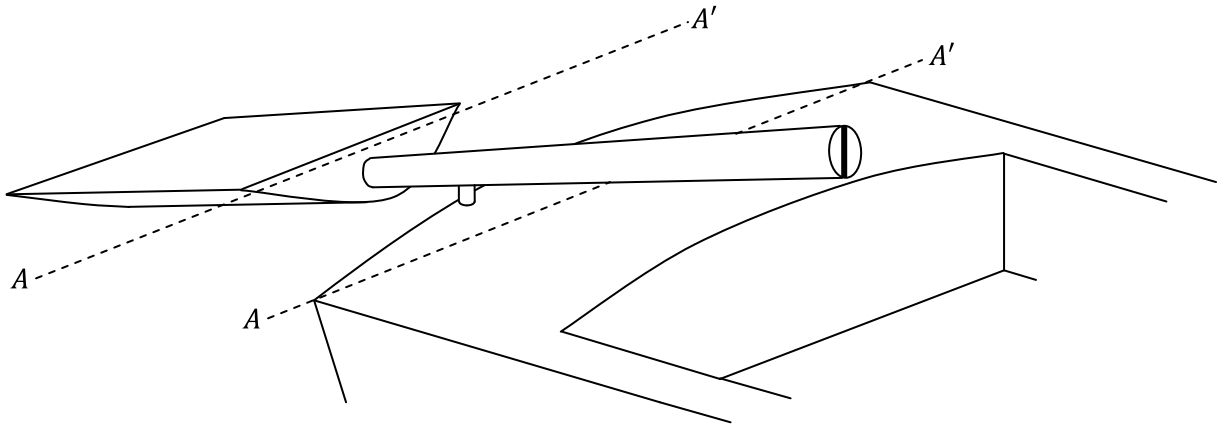
During the return stroke, the green arrows would be unchanged, but the red and black arrows would point generally downwards. But the magnitudes and angles of the relative "wind" would be the same.

These negative angles of attack are going to be a problem. There is no airfoil shape which will generate lift when the angle of attack is negative 30° or 40°. The solution is to rotate the airfoil around its long axis, which points vertically down into the water. It the angle of incidence is set to say, 45°, in the previous figure, then the angles of attack would vary between 45° - 27.9° = 17.1° at the start of the stroke to 45° - 44.1° = 0.9° at the end. Of course, this angle of incidence only works when the blade is traveling clockwise around the fulcrum, as shown in the previous figure. On the return stroke, the rotation of the blade would have to be reversed by 90°, so the angle of incidence is set to 45° with respect to the counter-clockwise sweep.

If Isambard Kingdom were to attack this problem, he would say, "No problem, I can gear this thing." Indeed, it would be possible to set up a gearing system. It would set the angle of incidence to 45° with respect to the long axis of the airfoil when the scull is sweeping in one direction and set it to 45° the other way when the scull is sweeping in the other direction. (One can get creative. A small vane placed somewhere on the blade could be introduced to measure the water direction; the gear train could be engineered to set the angle of incidence continuously with respect to the measured water direction. Naturally, GPS velocity-tracking of the airfoil and a fly-by-wire hydraulic system would be available as upgrades on more advanced models.)

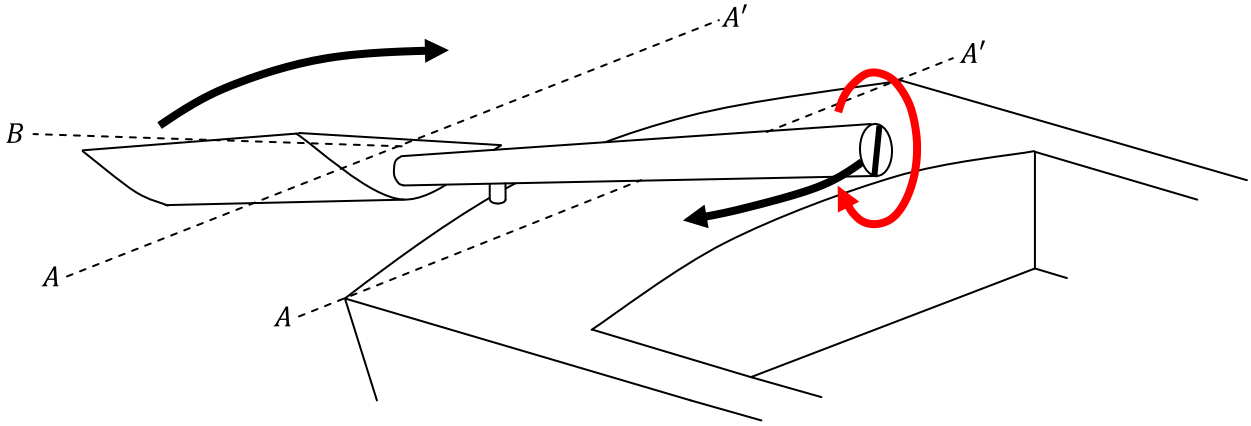
If these ±45° angles of incidence remind one of a typical boat's propeller, it is no accident. A typical propeller rotates around its shaft in a constant direction. It follows that the blades can be set at fixed angles. The angle of incidence will be optimal for only one forward speed of the boat. Presumably, the designer set the angle of incidence so the propeller would be most efficient at the speed at which the boat normally cruises. The airfoil we have been looking at seems to be a propeller consisting of a single blade. Furthermore, the airfoil does not always travel in the same direction, but reverses direction twice per stroke.

The traditional yuloh is an ingenious solution to the angle of attack problem. Start by angling the airfoil. Instead of pointing straight down, the long axis slants both downwards and rearwards from the aft deck. In the following diagram, I have shown a yuloh mounted on the transom of a dinghy. The scull is shown in its amidships position, with the flat rear side of the blade oriented parallel to a horizontal line across the aft deck. The two lines AA' are parallel. I have shown the fulcrum, or pivot point, a little bit to starboard, to make room for the operator, who stands in the cockpit a little bit to port. With the fulcrum to starboard, the operator would face the starboard side when sculling.

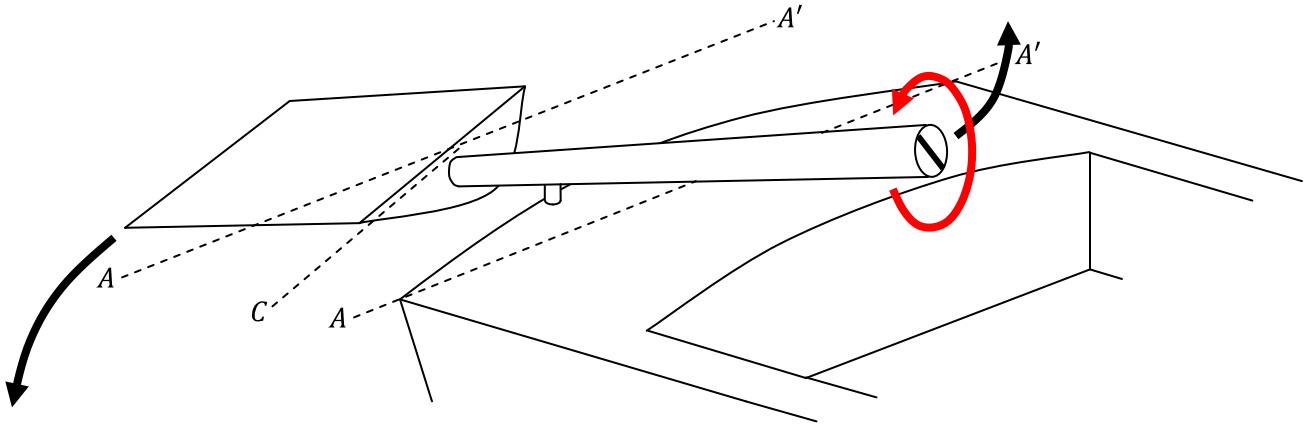


For reference in the following diagrams, I have shown a small index mark on the end of the loom. The index mark is perpendicular to the flat side of the side.

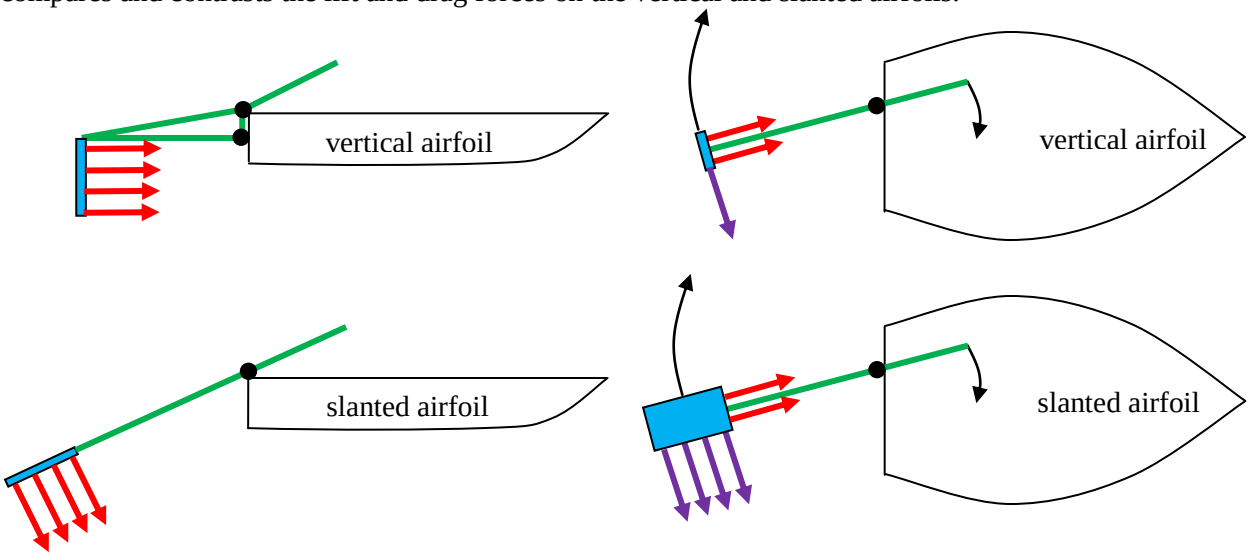
The ingenuity of the yuloh is this: it is rotated around its long axis to produce the required angle of incidence. The following figure shows the yuloh when the loom is being pushed to starboard, thus pulling the blade to port. The black arrows show the directions in which the ends of the yuloh are travelling. In addition, the Yuloh has been rotated around its long axis in the circular direction shown by the red arrow. Rotation cause the flat rear side of the blade to tilt along line B. This gives the blade a positive angle of attack, biting into the water, as it sweeps to port.



On the reverse stroke, the yuloh is rotated in the reverse direction, as the red arrow and index mark show in the following diagram. The rear flat side of the blade is angled along line C and encounters the water with an angle of attack (approximately) given by the angle between line AA' and line C.



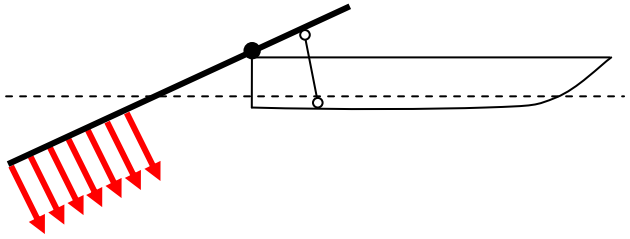
It is true that sloping the scull aft as well as down causes a loss of effectiveness. The following figure compares and contrasts the lift and drag forces on the vertical and slanted airfoils.



The two diagrams on the top row show the airfoil in a vertical orientation, with gearing of course. The lift generated by the blade (red arrows) is horizontal to the water's surface. Seen from above, the lift acts in a direction up the oar. Over the course of a stroke, or even a half-stroke, the average direction of the lift is the direction of the boat's velocity. The drag force (purple arrows) is a retarding force, and acts in the direction opposite to the airfoil's direction of travel.

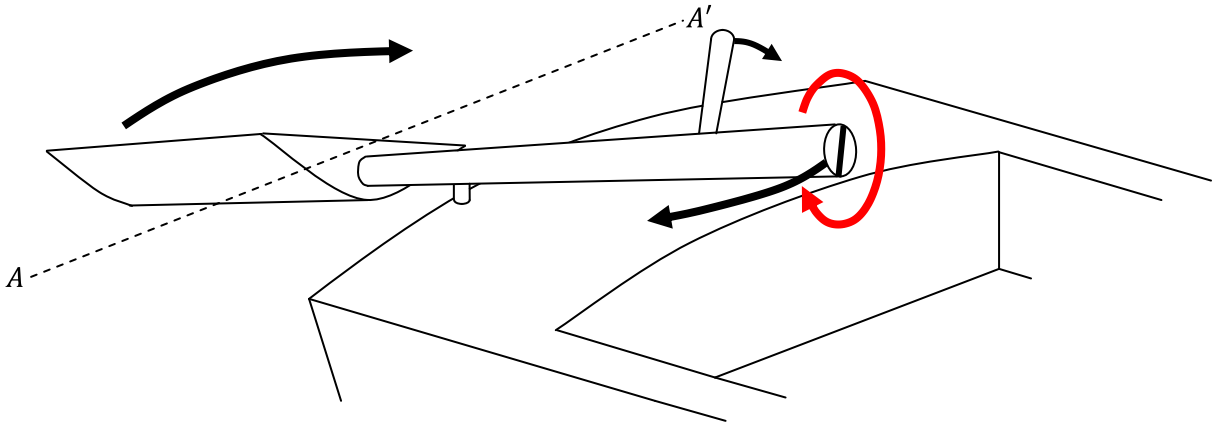
The lower row in the figure is a traditional yuloh. The lift forces (red arrows, again) have a downwards component as well. Unfortunately, the downwards component adds nothing to the forward progress of the boat. It just tends to push the stern deeper into the water. The drag forces (purple arrows, again) retard the progress of the airfoil in much the same way as before.

Increasing the steepness of the slant of the yuloh is a good thing. A steeper slope increases the forward component, and reduces the downward component, of the lift generated by the airfoil. On the other hand, making the yuloh steeper tends to raise its inboard end. The ideal height of the inboard end is (arguably) the narrow band between the operator's sternum and the bottom of his elbows. This height allows the operator to lean partially on the loom and to use his body weight, and not just his arms, to push the loom during the "push" half-stroke. Increasing the slant steepness can raise the inboard end to a height which is uncomfortable. A solution sometimes seen is the use of a "downhaul", for lack of a better word, something like that shown in the following figure.



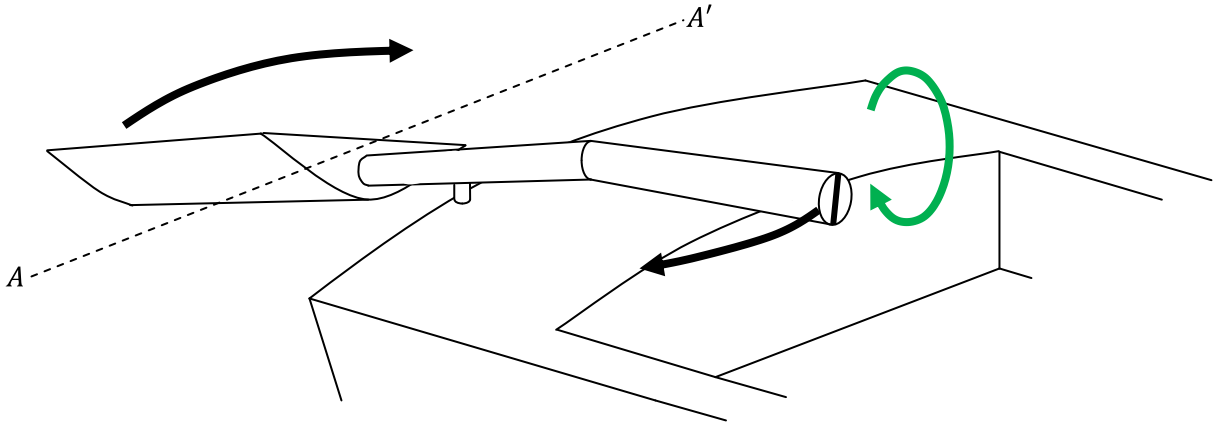
I have shown the downhaul secured to an eye bolt on the loom, at a station aft of the operator, and to an eyebolt on the keel or deck. The downhaul resists the downward component of the lift (red arrows, once again) generated by the blade. With a downhaul deployed, the operator need not press down on the loom. He only needs to sweep it back and forth. Use of a downhaul forces the yuloh to sweep out a section of a cone as it moves. This will cause the tip of the blade to move in circular arcs, both as seen from above and as seen from astern. As the blade sweeps through the water, its tip will not maintain a constant depth. I do not know if this is useful or not.

A small stick is often added to the loom, as is shown in the following figure.



The stick is typically six or eight inches long, and projects at right angles to the loom just at the station where the operator's aft hand grasps the loom. The stick is set into the shaft so that it is also perpendicular to the flat rear side of the blade, and thus parallel to the index marks shown before. The operator uses the stick to control the angle of incidence he sets on the blade. I have set up the figure so the loom is moving to starboard, the tip of the blade is moving to port and the required angle of incidence is set by twisting the loom angularly in the direction of the red arrow. The top of the stick needs to be angled in the direction opposite to the direction in which the inboard end of the loom is moved. In practice, operation of the stick comes naturally. It does not need hard pulls or shoves. Whatever the operator does with his fore hand, he has to do a little less of that with his aft hand. If his fore hand is pushing the loom, for example, all he needs to do is push a little less hard with his aft hand and the airfoil will take on the appropriate set. Typically, the operator grasps the loom with his aft hand in such a way that the stick projects up through the space between the thumb and the curled fingers. A little pressure one way or the other is all it takes.

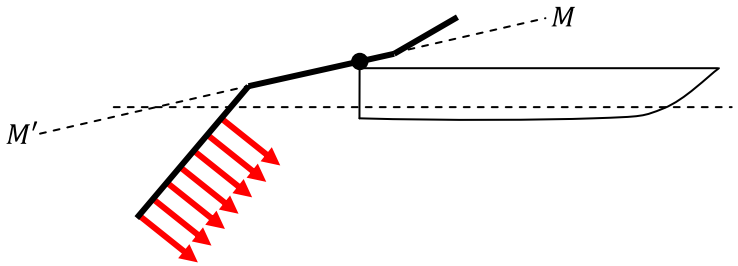
Another method is sometimes used to control the angle of incidence. It involves a small droop of the inboard end of the loom, as shown in the following figure.



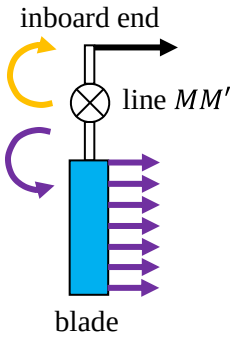
This figure shows the same half-stroke as the previous one, with the tip of the blade moving to port. When the operator pushes or pulls the inboard end to starboard, the droop causes a torque to be exerted on the loom around the axis which extends throughout its main length. Because the inboard end lies below the main axis of the loom, the force tends to rotate the loom in the angular direction shown by the green arrow. This is exactly the direction which will set a positive angle of incidence for this half-stroke.

The droop has a subtle, and helpful, side effect. The droop lowers the inboard end of the loom. To the extent that it is desirable to set the height of the inboard end at a particular height on the operator's torso, a droop allows the slant of the outboard end of the yuloh to be steepened. We saw above that such a steeper slant will direct a greater proportion of the lift in the useful direction.

Another variation sometimes seen is an S-shaped yuloh, as is shown from the side in the following figure. I have shown a dotted line MM' through the central axis of the middle section of the yuloh, to which I will refer in a moment.



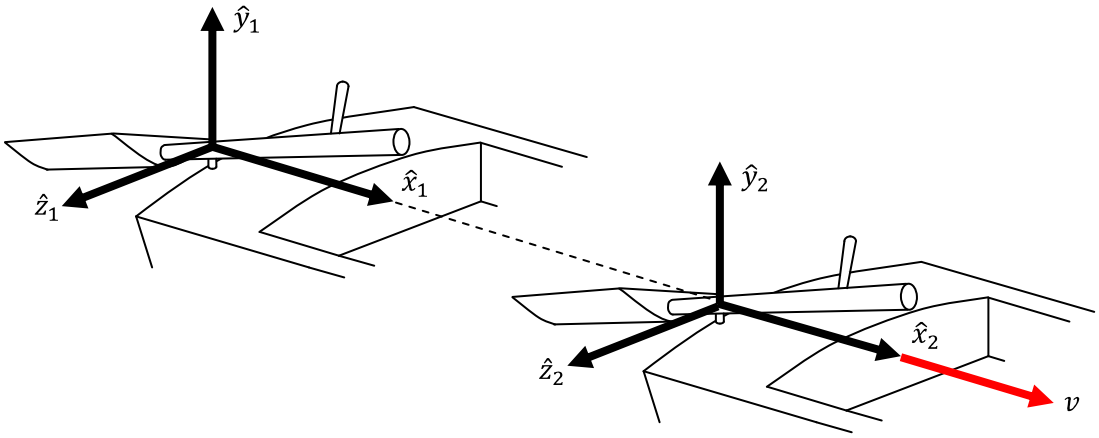
The S-shape is such that the slant of the blade is steeper than it otherwise would be. This is a good thing from the point-of-view of efficiency. But, it has a consequence which can be corrected by an upward slant, or anti-droop, at the inboard end. The following figure is a view of the scull looking down line MM' from the inboard end, point M.



As before, the airfoil itself is shown in blue. The drag forces acting on the airfoil are shown by the purple arrows. The black arrow shows the direction in which the operator is pulling the inboard end of the loom. The blade is, of course, moving in the opposite direction, so the drag forces act in the same direction as the force on the inboard end of the loom. The drag forces exert a counter-clockwise torque around line MM', in the angular direction shown by the purple semi-circle. Because of the anti-droop on the inboard end, the force on the inboard end exerts a clockwise torque around line MM', in the angular direction shown by the orange semi-circle. If the amount of anti-droop is correct, the two torques will cancel each other out, so there will be no net torque tending to change the angle of incidence.

A mathematical expression for the angle of attack

It is useful to find a mathematical expression for the angle of attack as the blade moves through the water. This is most easily done using a sequence of frames of reference. Let's begin with a frame of reference which is stationary with respect to the water. I will call this the $\hat{1}$ frame of reference and define its three axes as shown in the following figure. I will call the three axes the $\hat{x}_1$, $\hat{y}_1$ and $\hat{z}_1$ axes, respectively, where the subscripts on the axes' symbols tie them to their frames of reference. The following figure also shows the $\hat{2}$ frame of reference. I will describe the relationship between these two frames of reference in a moment.



The $\hat{1}$ and $\hat{2}$ frames of reference both have their origins at the center of the fulcrum, which is assumed to lie on the central axis of the yuloh. The $\hat{y}_1$ and $\hat{y}_2$ axes are both perpendicular to the water surface. We will ignore all changes in the boat's attitude -- rolling, pitching and yawing -- and assume the boat is

sailing straight on a calm pond. In both frames of reference, the $\hat{z}$-axis points due starboard. And, in both frames of reference, the $\hat{x}$-axis is parallel to the boat's centerline. Since the $\hat{y}$ axes are perpendicular to the water surface, the $\hat{x}$ and $\hat{z}$ axes define planes which are parallel to the water's surface and a constant distance above it.

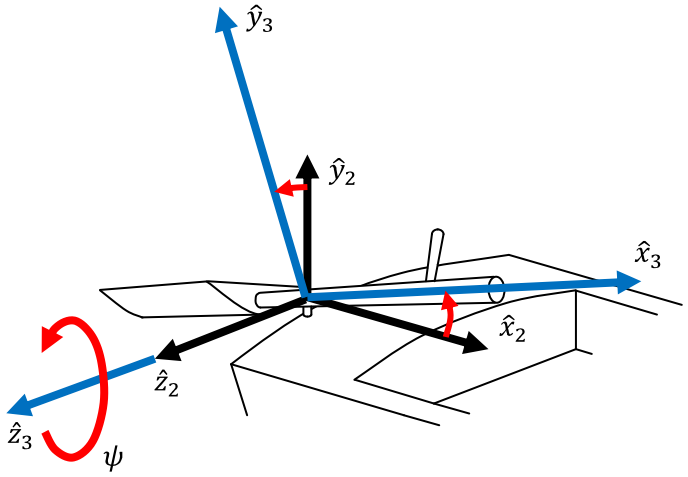
The difference between the two frames of reference arises because the boat moves at a constant speed v out along the $\hat{x}_1$ axis. The $\hat{2}$ frame of reference has its origin fixed to the fulcrum. It is a "boat-fixed" frame of reference. But, the $\hat{1}$ frame does not move. It is fixed with respect to the water. For our purposes, we will say that the origin of the $\hat{1}$ frame of reference remains located at the point the fulcrum occupied at time $t = 0$, at which time we will start taking measurements.

I have defined these two frames of reference, and will define a few more below, because they are convenient ways in which to describe the exact location of a point. For example, the operator's sternum might be located four feet in front of the fulcrum, one and one-half feet higher and, perhaps, two feet to port. In feet, the co-ordinates of his sternum could be written as (4, 1.5, -2), where the three numbers are the distances in each of the three directions, given in the order x-y-z. Actually, these are the co-ordinates of his sternum only in the boat-fixed frame of reference. From the point-of-view of the $\hat{1}$ frame of reference, the operator is moving continuously down the $\hat{x}_1$ axis. The $\hat{x}_1$ co-ordinate of the operator's sternum may have been four feet at time $t = 0$, but, as time passes, that co-ordinate increases. If the speed of the boat v is measured in feet per second, then after t seconds, the boat will have moved a distance vt, in feet. The co-ordinates of the operator's sternum in the $\hat{1}$ frame of reference is (4 + vt, 1.5, -2). As an aside, note that the co-ordinate in the $\hat{z}$-direction is algebraically negative, which simply means that the distance of two feet is to be taken in the direction of the negative $\hat{z}$-axis, namely, to port.

In general, if the location of a particular point has the co-ordinates (x_2, y_2, z_2) in the $\hat{2}$ frame of reference, its co-ordinates expressed in the $\hat{1}$ frame of reference are the following:

$$\left.\begin{aligned} x_1 &= x_2 + vt \\ y_1 &= y_2 \\ z_1 &= z_2 \end{aligned}\right\} \quad (1)$$

The following figure shows how I will define the $\hat{3}$ frame of reference. Starting with the $\hat{2}$ frame of reference, we will make a rotation of angle ψ -- the Greek letter "psi" -- around the positive $\hat{z}_2$ axis. For the sake of clarity, I have shown the axes of the $\hat{3}$ frame of reference in blue and longer than their counterparts in the $\hat{2}$ frame. The direction of rotation is shown by the red partially-elliptical arrow.

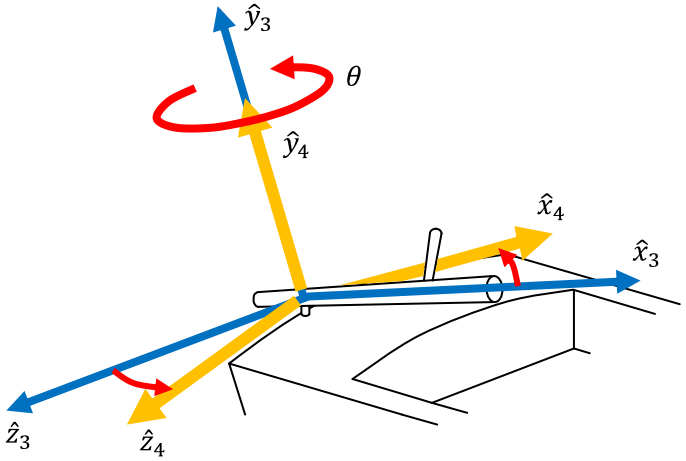


The amount of rotation is chosen so that the $\hat{x}_3$ axis is coincident with the centerline of the yuloh when it is positioned fore-and-aft. Angle ψ is therefore the angle by which the yuloh is slanted, to use the same term I used above. Since the rotation occurs around the $\hat{z}$ axis, it does not change the z-co-ordinate. Only the $\hat{x}$ and $\hat{y}$ values are changed by the rotation.

The transformation which relates the co-ordinates of any particular point in these two frames of reference is a little more complicated than for the translation which transformed the $\hat{1}$ frame of reference into the $\hat{2}$ frame. It is this:

$$\left.\begin{aligned} x_2 &= x_3 \cos\psi - y_3 \sin\psi \\ y_2 &= x_3 \sin\psi + y_3 \cos\psi \\ z_2 &= z_3 \end{aligned}\right\} \quad (2)$$

The transformation from the $\hat{3}$ frame of reference to the $\hat{4}$ frame of reference takes into account the side-to-side sweep of the yuloh. The $\hat{3}$ frame of reference was defined with the yuloh held precisely fore-and-aft. We will represent the side-to-side deflection by the angle θ -- the Greek letter "theta" -- which the loom makes with respect to the $\hat{y}_3$ axis. The following figure shows the relationship between the $\hat{3}$ and $\hat{4}$ frames of reference. For the sake of clarity, I have shown the axes of the $\hat{4}$ frame of reference in orange and shorter than their counterparts in the $\hat{3}$ frame. I have also omitted the airfoil from the figure. The direction of rotation by angle θ is shown by the red partially-elliptical arrow.

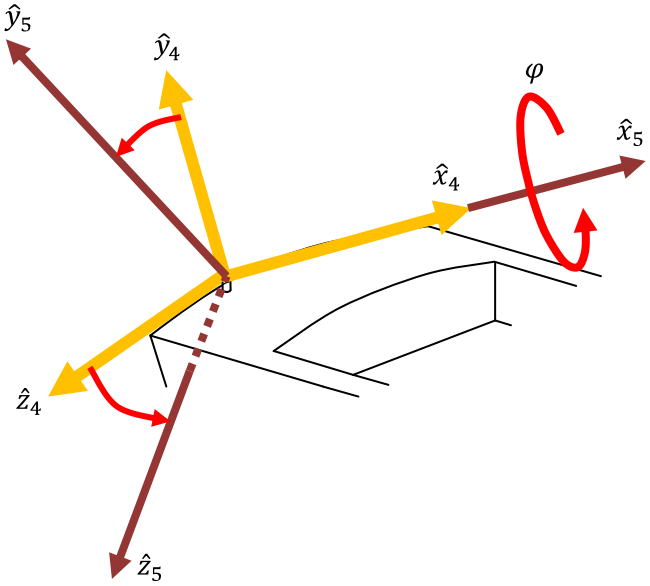


By representing the side-to-side motion of the yuloh by a rotation around the $\hat{y}_3$ axis, I have made the implicit assumption that the yuloh is constrained to lie entirely within the $\hat{x}_3$-$\hat{z}_3$ plane. This means that the inboard end of the scull will not remain at exactly the same height above deck. It moves in a little hump as it crosses from one side to the other. Similarly, the tip of the blade does not remain exactly the same distance underwater. It travels in a little trough as it moves from one side to the other. Other assumptions could be made. For a preliminary analysis, the one I have made here is satisfactory.

The transformation which relates the co-ordinates of the $\hat{4}$ frame of reference to those in the $\hat{3}$ frame of reference is similar to its predecessor. It is this:

$$\left.\begin{aligned} x_3 &= x_4 \cos\theta + z_4 \sin\theta \\ y_3 &= y_4 \\ z_3 &= -x_4 \sin\theta + z_4 \cos\theta \end{aligned}\right\} \quad (3)$$

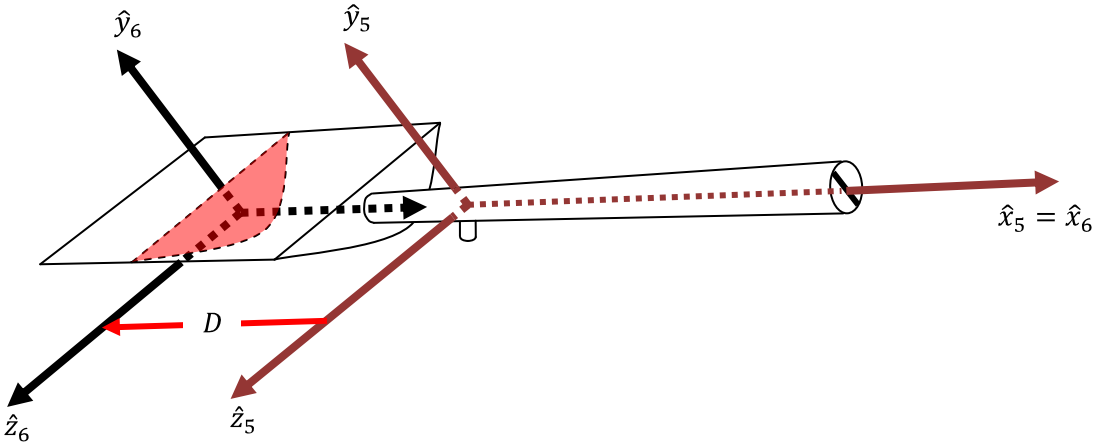
Things have been defined so far in such a way that the $\hat{x}_4$ axis is always coincident with the central axis of the yuloh. In the next transformation, we will rotate the yuloh around its long axis. We will use the symbol φ -- the Greek letter "phi" -- for the angle of rotation. To keep track of the direction of rotation, we will assume that angle φ is algebraically positive for rotations clockwise around the $\hat{x}_4$ axis. The following figure shows the setup.



For the sake of clarity, I have omitted all of the yuloh, leaving behind only the fulcrum. The $\hat{5}$ frame of reference is fixed to the yuloh. One should think of the $\hat{y}_5$ axis as being perpendicular to the flat rear side of the blade. This rotational transformation has the following form:

$$\left.\begin{aligned} x_4 &= x_5 \\ y_4 &= y_5 \cos\varphi - z_5 \sin\varphi \\ z_4 &= y_5 \sin\varphi + z_5 \cos\varphi \end{aligned}\right\} \qquad (4)$$

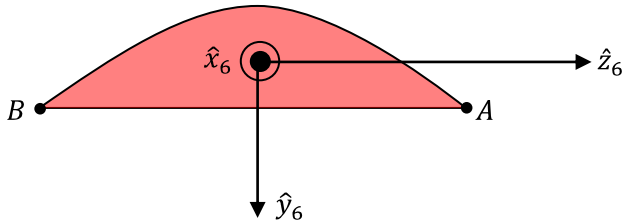
The next transformation is the last we will consider. It is a translation of the $\hat{5}$ frame of reference down the shaft of the yuloh to some particular cross-section of the foil. I will use the symbol D for the distance the frame of reference is moved. By letting D vary, we can select different sections along the airfoil. The following figure shows the translation.



The transformation can be written algebraically as follows:

$$\left.\begin{aligned} x_5 &= x_6 + D \\ y_5 &= y_6 \\ z_5 &= z_6 \end{aligned}\right\} \quad (5)$$

The cross-section of the blade at displacement D is highlighted in light red in the figure above. The $\hat{6}$ frame of reference is ideal for use in describing points on the surface of the blade. The highlighted coss-section is shown again in the following figure. The $\hat{x}_6$ axis points directly out of the page. The $\hat{z}_6$ axis is parallel to the flat rear side of the blade and the $\hat{y}_6$ axis points straight out of the flat side. It is likely that the loom pierces the cross-section at its centroid.



I have identified two points, A and B, about which I will talk some more below. These two points are the leading and trailing edges of this cross-section of the airfoil. Neither one is strictly "leading" or "trailing"; they exchange roles every half-stroke.

For illustration, let's assume that the blade has been shaped from a piece of standard 2" by 4" lumber. The true dimension of stock 2×4 are 1½ inches and 3½ inches, respectively. The centroid of this cross-section will be located at the mid-point widthwise and about one-third of the distance from the bottom to the top. In inches, the co-ordinates of point A are $(0, -½, 1¾)$. Point B is a mirror image and will have co-ordinates $(0, -½, -1¾)$. Expressed in feet, the co-ordinates in the $\hat{6}$ frame of reference of the two points are $(x_6, y_6, z_6) = (0, -0.0417, \pm 0.146)$. I have chosen to express these distances in feet, because we already used feet to measure the boat's speed and to estimate the yuloh's length.

We can express the co-ordinates of these two points in the $\hat{1}$ frame of reference by applying the transformations one after the other. It is easiest to write the result as a sequence of matrix multiplications, as follows:

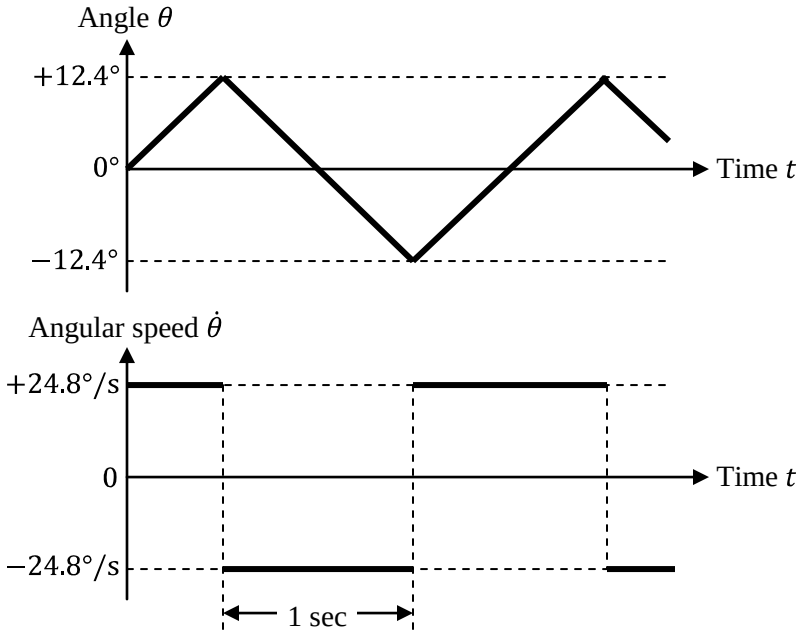
$$\begin{aligned}
\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} &= \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} + \begin{bmatrix} vt \\ 0 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} + \begin{bmatrix} vt \\ 0 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} x_4 \\ y_4 \\ z_4 \end{bmatrix} + \begin{bmatrix} vt \\ 0 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} x_5 \\ y_5 \\ z_5 \end{bmatrix} + \begin{bmatrix} vt \\ 0 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} x_6 + D \\ y_6 \\ z_6 \end{bmatrix} + \begin{bmatrix} vt \\ 0 \\ 0 \end{bmatrix}
\end{aligned} \quad (6)$$

Little is to be gained by multiplying the expression out by hand. I believe it is more informative to see how points A and B move through time when seen from the $\hat{1}$ frame of reference. Recall that the $\hat{1}$ frame is in a fixed position with respect to the water. Its origin is the point occupied by the fulcrum at time $t = 0$. We already have the co-ordinates of the two points, which are the fixed values $(0, -0.0417, \pm 0.146)$ we wrote down a paragraph or two ago.

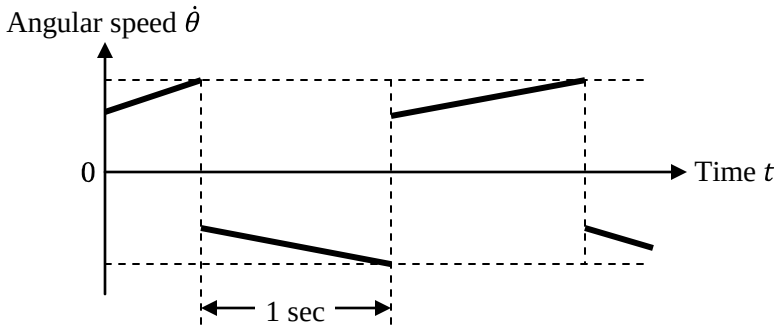
For our numerical example, we will use the same boat speed we used at the beginning of this paper: 1½ knots, so $v = 2.53$ feet per second. We will look at a cross-section of the blade which is a distance eight feet down the shaft from the fulcrum. This should be near the center of the airfoil's long length. We will substitute $D = 8$ feet into Equation (6).

Angle ψ is the slant angle of the Yuloh. In our mathematical model, ψ does not change with time. Let's assume that the slope is 40°.

The remaining two variables are angles θ and φ. Both vary with time. θ is the angle through which the yuloh travels from side-to-side. In the analysis of the rudder-type yuloh, we estimated that the yuloh moved was pushed or pulled to a maximum deflection angle of 12.4°. We also assumed the operator ran at 30 strokes per second. We will use the same maximum deflection and frequency in this example. We also have the opportunity to select a stroke pattern. I am going to assume that the operator moves the inboard end with a constant angular speed, first in one direction and then the other. A constant angular speed means that the deflection angle θ describes a sawtooth waveform as a function of time. The following figure shows the angle and angular speed of this stroke pattern. The angular speed is 24.8 degrees per second in one direction, followed by 24.8 degrees per second in the other.

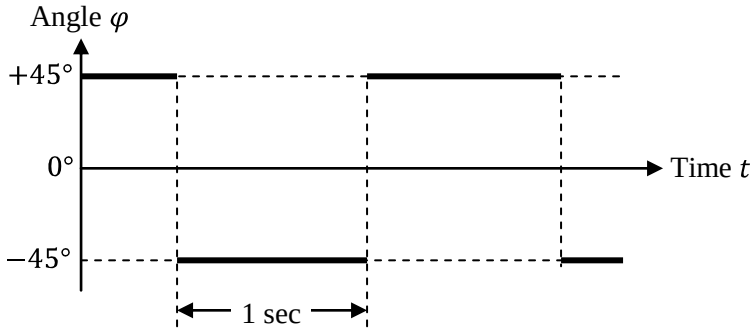


The stroke pattern is not sinusoidal, nor is it intended to be. I believe the sawtooth waveform better represents the constant speed of a typical operator. In fact, I believe there is an even better representation, but one which is better left for a later analysis. A typical operator applies a constant force to the loom. This does not necessarily result in a constant sweep speed. We saw above that the angle of attack with which the blade meets the water is highest at the start of a half-stroke and lowest at the end. To the extent that the drag force is proportional to the angle of attack, the loom will be hardest to move at the start of the stroke. A constant exertion on the loom will likely cause the yuloh to start off slow and then to speed up during the half-stroke. A more realistic stroke pattern is likely the following:



For the numerical example, we will use the constant angular speed version, and defer any enhancements.

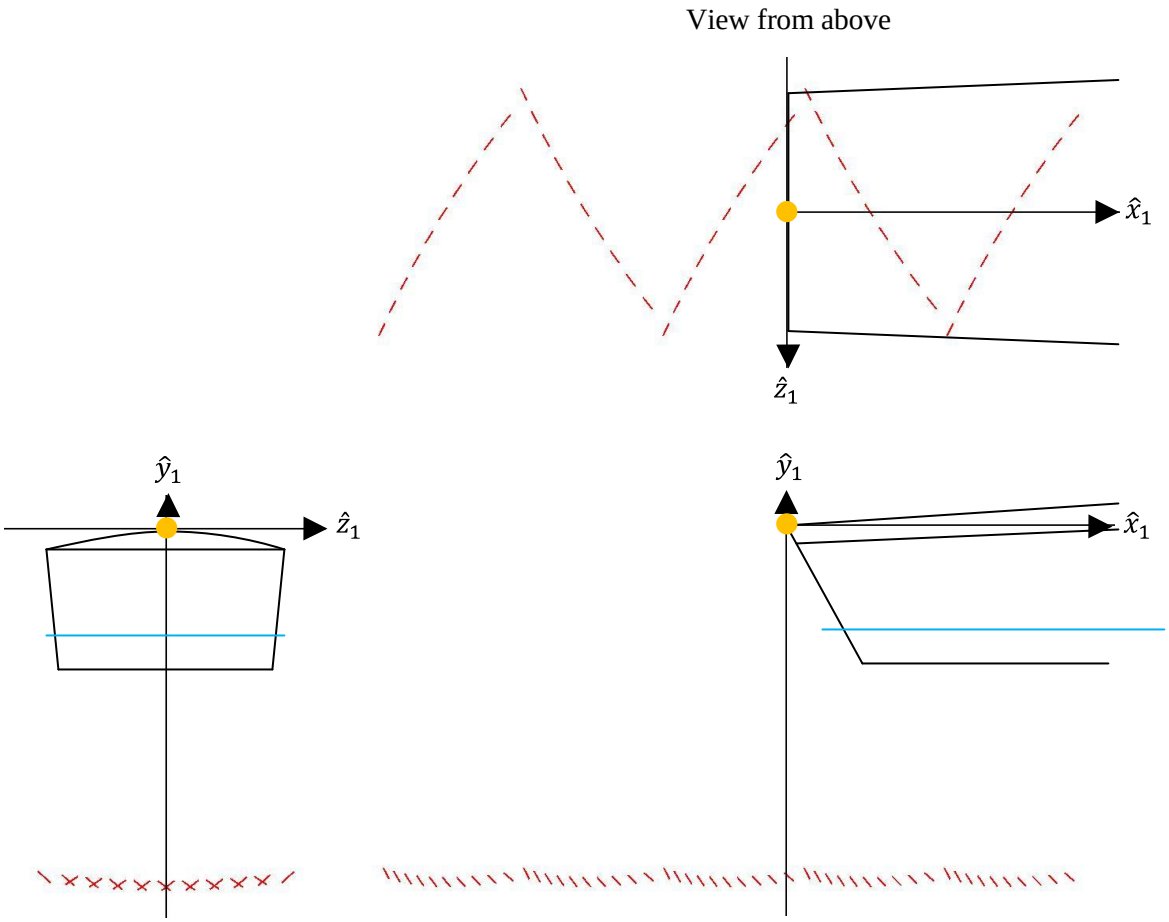
The last remaining variable is the loom twist angle φ. We will assume this angle is held constant during a half-stroke and then reversed during the following half-stroke. I proposed above that the angle of incidence could be 45°, and that is the value I will use in the numerical example. The following figure shows more precisely the waveform assumed for φ in the numerical example.



Before proceeding, I want to make sure that angles θ and φ are algebraically consistent. The graphs in the figures show that angle φ is positive (or negative) when the angular speed $\dot{\theta}$ is positive (or negative). We need to confirm that this is the correct combination. Look back at the figure which shows the rotation of the $\hat{3}$ frame of reference into the $\hat{4}$ frame of reference. An increasing angle θ corresponds to the tip of the blade moving towards the starboard side. Now, look at the figure which shows the rotation of the $\hat{4}$ frame of reference into the $\hat{5}$ frame of reference. A positive angle φ causes the starboard edge of the blade to descend and the port side to ascend. This is exactly what we want -- the starboard edge lower than the port edge when the blade is moving to starboard.

The following figure is a 3,000 word essay on the trajectory which the line segment from point A to point B makes during five half-strokes starting at time $t = 0$ with the tip of the blade beginning a sweep to port. The three diagrams show the top view, the side view and the rear view. The line segment $\overline{AB}$ is rendered in red and is shown every 0.1 seconds, or ten times every half-stroke. I have not labeled the axes with dimensions in feet, but all three views are to the same scale.

In all cases, the two axes in the plane shown intersect at their origin. When interpreting the views, remember that the origin of the $\hat{1}$ frame of reference is the location of the fulcrum at time $t = 0$. I have marked this location with an orange dot in the views. This location is, of course, some distance above the waterline. Also bear in mind that the yuloh projects aft of the transom. Notice that the line segment does not pass through the $x_1 = 0$ datum (the starting line, if you will) until almost the end of the third half-stroke. The view from above is sometimes called the "falling leaf" pattern, for obvious reasons.



The objective of the analysis at hand is not to draw pretty pictures of the blade passing through the water, but to estimate the angle at which water attacks the blade. Fortunately, these two objectives are opposite sides of the same coin. If we can determine the path the blade takes through the water (as we have just done), then, from the point-of-view of the blade, the water is approaching from the opposite direction at the same angle(s).

It should be understood that the flow of water over the blade is very complex. It has complications not found in the preliminary analysis of an airplane's wing in cruising flight. Not only are different sections along the blade's long axis moving through the water at different speeds, but the long axis is rotating with respect to the water as well. It is not possible to ascribe a single angle of attack to the yuloh blade, even at a single instant in time. If we are to make any progress at all in understanding the hydrodynamics, we are going to have to make some rather crude space-averaged or time-averaged assumptions. Here is how I have chosen to proceed.

I have assumed that the blade of the yuloh is five feet long and that the section $\overline{AB}$ we looked at in the immediately preceding graphical example, at displacement $D = 9$ feet, was located at the mid-point of the long axis of the blade. In other words, the blade extends from $D = 6½$ feet to $D = 11½$ feet down the shaft from the fulcrum. I have chosen to focus on only 11 points along the long axis of the blade, being values of D one-half foot apart from $D = 6½$ feet to $D = 11½$ feet, inclusive.

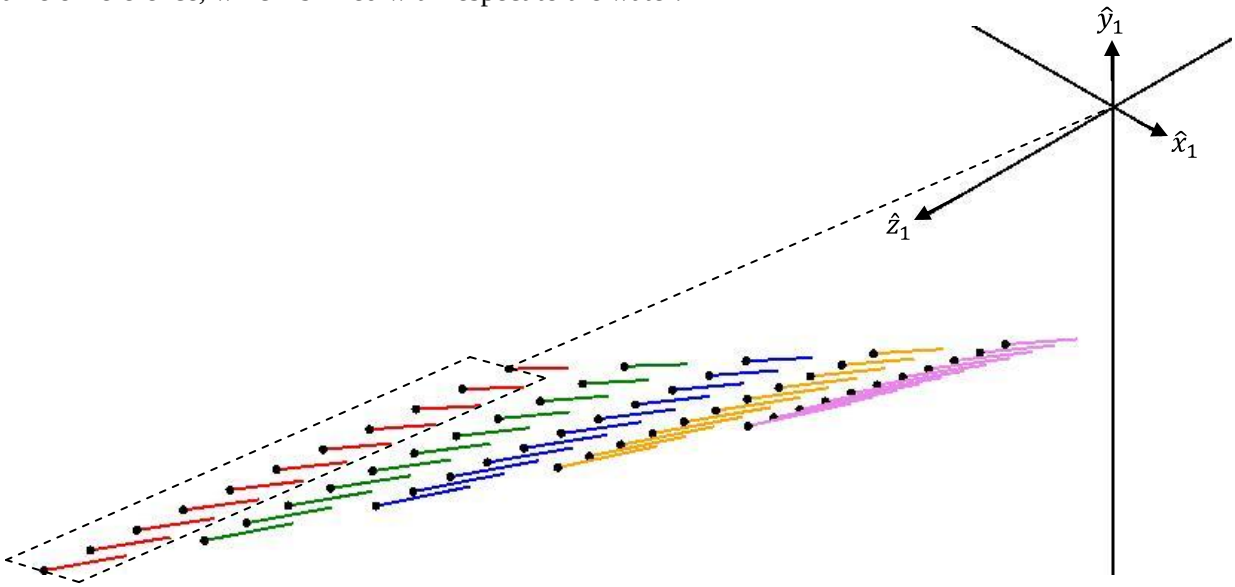
For each of these 11 cross-sections of the blade, I have chosen to look only at the relative speed with respect to the water at the centroid of the section. For this purpose, I have assumed that the points on the $\hat{x}_6$ axis lie on the cross-sectional centroids of the profile. These points have co-ordinates $y_6 = z_6 = 0$. Ironically, this means that none of the 11 points at which I will calculate the angles of attack actually lie on the surface of the blade; they are in its interior.

For each of these 11 points, I have chosen to look at the relative speed with respect to the water at five equally-spaced times during a single half-stroke. It makes no difference whether we consider a stroke to port or one to starboard, since the effects we will be looking at are symmetric about the centerline.

All told, I will be calculating 55 angles of attack. To actually calculate the relative speeds, I have chosen to use an expeditious method. Rather than apply the Calculus to Equation (6) to calculate the relative speed in closed form, I have used computed differences. At five selected times during the half-stroke, I calculated the positions of the 11 points. I then calculated the 11 positions again at a time one millisecond later. The change in position, divided by the one millisecond change in time, gives the average speed during the interval. Actually, it gives the components of the average speed along all three axes, in other words, the velocity. If we keep track of the co-ordinates of the 55 data points at the start and end of the 0.001 second interval, we can calculate the velocity in either the $\hat{1}$ or $\hat{6}$-frame of reference. In the course of this work, we will have to transform points from the the $\hat{1}$-frame of reference back to the $\hat{6}$-frame, in the direction opposite to the transformation in Equation (6). The inverse of Equation (6) can be written down by inspection, since the inverse of a rotation matrix is merely its transpose. We get:

$$\begin{bmatrix} x_6 \\ y_6 \\ z_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi \\ 0 & -\sin\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 - vt \\ y_1 \\ z_1 \end{bmatrix} - \begin{bmatrix} D \\ 0 \\ 0 \end{bmatrix} \quad (7)$$

The following graphs show the results for the same numerical values used to derive the falling leaf pattern. The first of the graphs shows the relative speed of the blade with respect to the water in the $\hat{1}$-frame of reference, which is fixed with respect to the water.

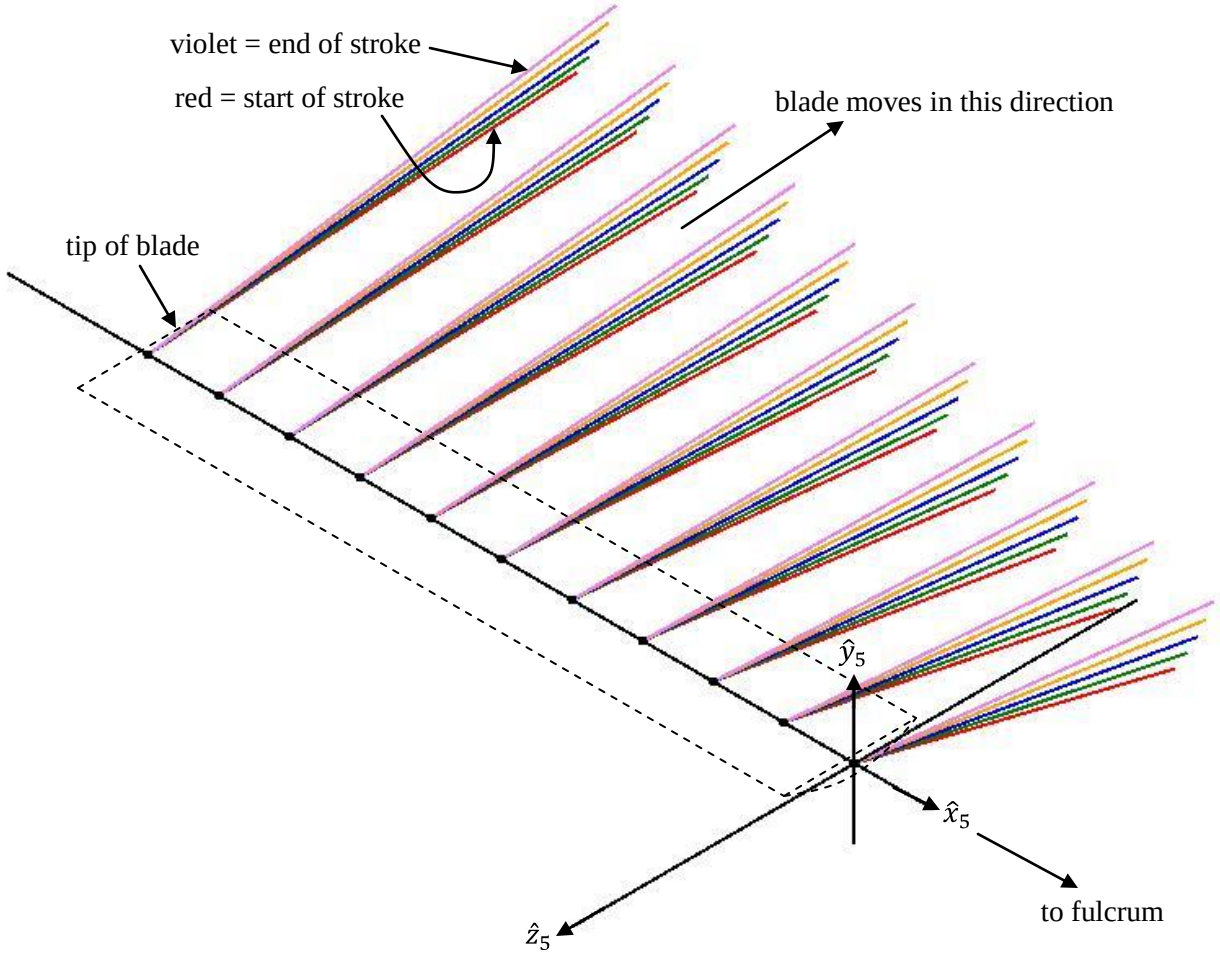


The dotted parallelogram is the outline of the blade at the start of the stroke. The red line segments have lengths which are proportional to the relative speed with respect to the water. The small black dots identify the locations of the 11 points along the long axis of the blade at the start of the one millisecond

interval. The red line segments apply at the start of the half-stroke when the yuloh is at its maximum deflection (12.4°) to starboard. One-quarter second after the start of the half-stroke, the yuloh is one-quarter of the way across its sweep, and the green line segments apply. The blue line segments apply at the mid-point of the stroke, when the yuloh is aligned fore-and-aft. The end of the half-stroke is represented by the violet line segments.

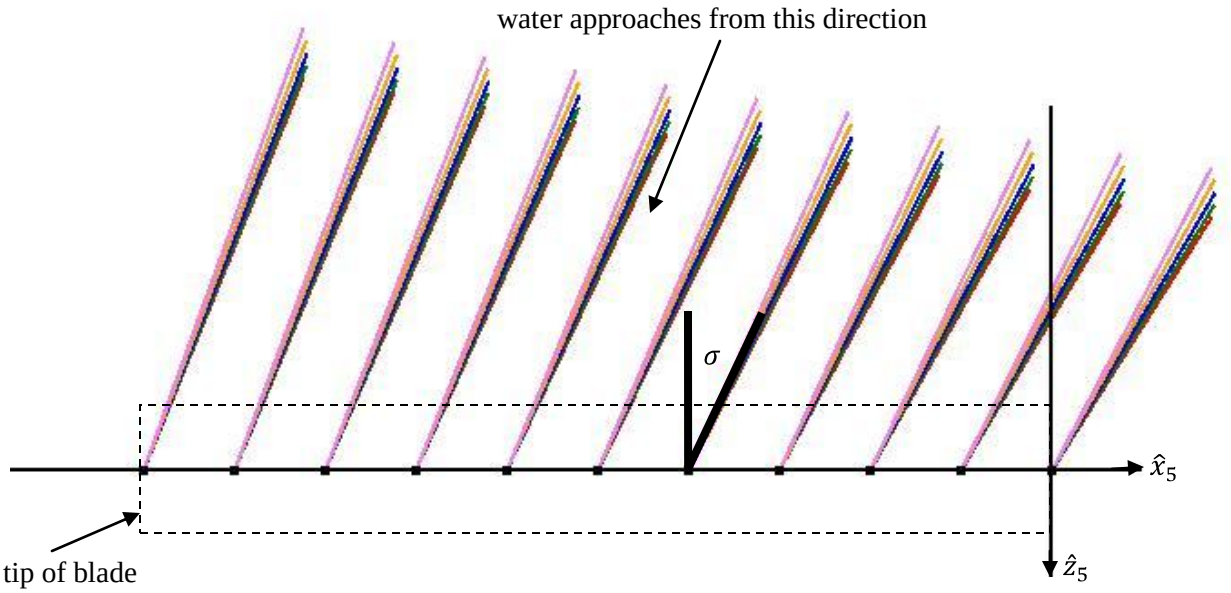
The $\hat{x}_1$-axis, in which direction the boat is travelling, points at a 30° towards the lower right. The $\hat{z}_1$-axis, directed due starboard, points at a 30° angle towards the lower left. The $\hat{y}_1$-axis is the vertical.

The next graph shows the relative speed once more, but this time in the $\hat{5}$-frame of reference. Both the $\hat{5}$-frame and the $\hat{6}$-frame are fixed to the blade of the Yuloh. The only difference between them is the distance along the long axis of the blade chosen as the origin of the latter frame of reference. The results are more informative if the starting points of the relative speeds (the small black dots) are shown separated along the blade in a realistic manner.



We are getting close now. The lines in this figure show the direction in which the blade is moving with respect to the water, starting at the black dots and progressing towards the upper right. The water is approaching the blade from the opposite directions. Information about the angles of attack is contained in this figure, but is partly obscured by the existence of a second angle, which I will refer to as the "streamline angle". In the standard preliminary analysis of an airplane wing, it is usually assumed that the flow of air is perpendicular to the long axis of the wing. That is not the case here. The water approaches the blade with a component of speed in the same direction as the long axis of the blade. The water does not flow over the blade directly from the leading edge to the trailing edge along the shortest path, but

takes an oblique path over the blade. To show this, the following figure is a projection of the line segments in the figure above onto the $\hat{x}_5$-$\hat{z}_5$ plane. It is a view of the situtaion as seen from above, looking down onto the flat rear side of the blade.



I have defined an angle σ -- the Greek letter "sigma" -- to represent the spanwise-angle at which the streamlines approach the blade. For the numerical example being tackled, σ is relatively constant along the blade. To be particular about it, the streamline angle is a little greater at the inboard end of the blade and decreases nearer the tip. The streamline angle is also greater at the start of a stroke (the red line segments) and decreases as the stroke progressess to its end (the violet line segments).

The following table sets out the values of σ at representative places and times.

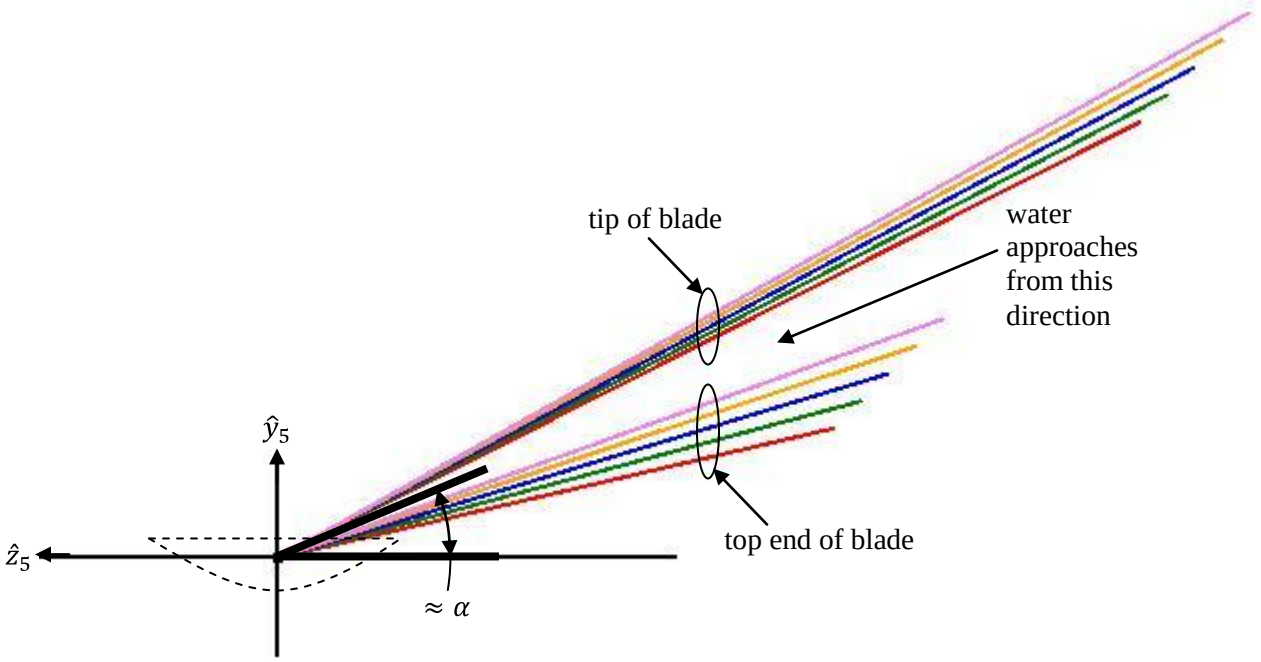
Angle σ, in degrees	**Upper end of blade**	**Halfway along blade**	**Tip of blade**
Start of stroke	32.3°	26.4°	22.2°
End of stroke	27.8°	23.2°	19.9°

The following figure is a projection of the line segments in the figure above onto the $\hat{y}_5$-$\hat{z}_5$ plane. It is a view of the situation as seen looking down the long axis of the blade. To avoid clutter, I have not shown the relative speed line segments for all 11 sections along the blade. I have shown only two sets of line segments, one at the top end of the blade and the other at the tip.

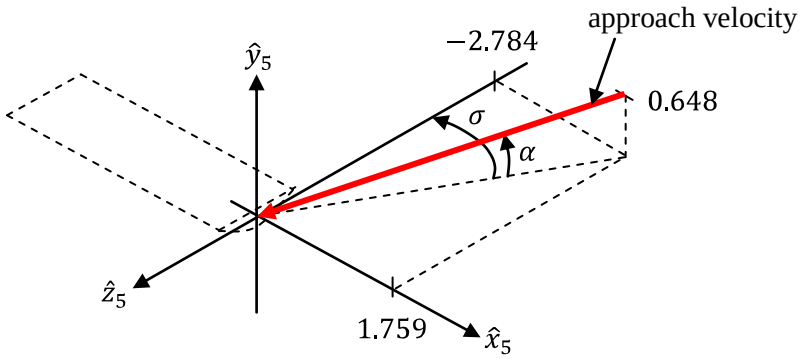
It is tempting, but incorrect, to define the angle of attack from what can be seen in the figure. To represent the angle of attack, I will use the symbol α, which is the Greek letter "alpha". It seems natural to use the flat rear face of the blade as the reference chord line. The angle of attack is the angle at which the oncoming water approaches the reference chord. However, the subtended angles shown in the figure are only approximately equal to α. I will explain why after a brief philosophical interlude

The angles of attack shown in the figure are useful for developing lift. That is, the water approaches the blade from the side "below" the curved surface. If we imagine the figue to be flipped top-to-bottom, the similarity to a traditional airfoil will be apparent. I observe that the angles of attack seem to be quite large

compared with those encountered in airfoil work. This arises because we set the loom twist angle to 45° in the numerical example. Perhaps this is too great an angle; we shall have to see.



Let me return now to calculating the angle of attack. The following figure shows the relative speed vector at only one of the 55 data points. For illustration, I will consider the approach vector at the top end of the blade and at the start of a half-stroke. The three components of the approach speed are identified in the figure.



The streamline angle σ and the angle of attack α are readily computed using the following trigonometry:

$$\alpha = \cos^{-1}\left(\frac{\sqrt{x_5^2 + z_5^2}}{\sqrt{x_5^2 + y_5^2 + z_5^2}}\right) \qquad (8A)$$

$$\sigma = \tan^{-1}\left(\frac{x_5}{-z_5}\right) \qquad (8B)$$

The following table sets out the values of α at representative places and times. For the sake of completeness, I have presented a second table as well, which sets out the relative speed between the blade and the water at the same points in space and time.

Angle α, in degrees	**Upper end of blade**	**Halfway along blade**	**Tip of blade**
Start of stroke	11.1°	19.6°	25.1°
End of stroke	17.6°	23.7°	27.8°

Speed, in fps	**Upper end of blade**	**Halfway along blade**	**Tip of blade**
Start of stroke	3.36 fps	4.21 fps	5.14 fps
End of stroke	3.95 fps	4.86 fps	5.84 fps

The angles of attack are higher at the tip than at the root of the blade. The angles increase as the stroke progresses. The relative speed is also higher at the tip and also increases as the stroke progresses.

We can control the angle of attack by setting different loom twist angles. For the sake of comparison, the following two tables set out the streamline angle and angle of attack for the same numerical example, with the exception of a loom twist angle of 35° instead of the 45° used above.

Angle σ, in degrees	**Upper end of blade**	**Halfway along blade**	**Tip of blade**
Start of stroke	31.6°	25.2°	20.8°
End of stroke	26.8°	21.9°	18.5°

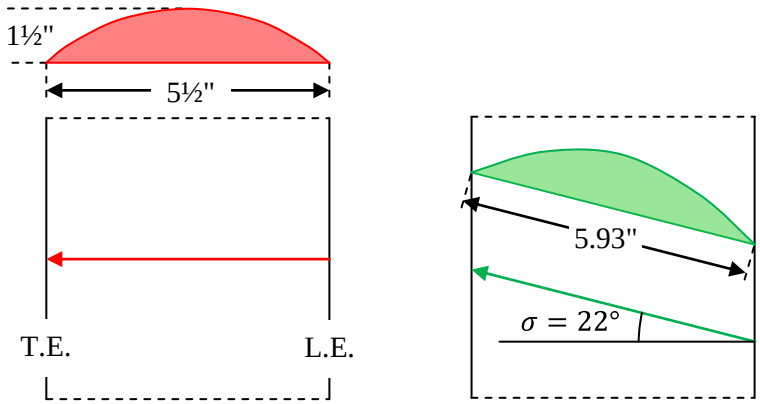
Angle α, in degrees	**Upper end of blade**	**Halfway along blade**	**Tip of blade**
Start of stroke	2.6°	10.6°	15.7°
End of stroke	8.7°	14.5°	18.3°

It should be noted that the streamline angles σ change by less than one and one-half degrees despite the ten degree reduction in the loom twist angle. The angles of attack α, on the other hand, are reduced by substantially all of the reduction in the loom twist angle.

This asymmetry suggests that a useful route in which to proceed is to assume a fixed streamline angle, say $\sigma = 22°$, and to investigate a range of angles of attack.

A preliminary look at the hydrodynamics of the yuloh

Fixing the streamline angle, at least initially, takes care of more than just one variable among many. It also sets the effective profile of the airfoil we will be testing. As a starting point, let's assume that the blade is shaped from a stock 2 × 6 piece of lumber. We will leave one side flat and shape the top into a segment of a circular cylinder. This shape is shown on the left in the following figure.

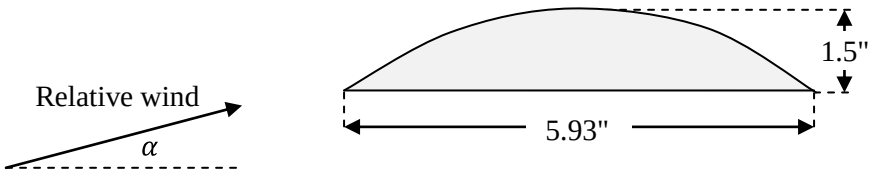


A piece of the blade is shown in top view beneath the profile. When a streamline of water approaches the blade at right angles, in the direction of the red arrow, the water must flow over/under a circular "bump" which is 1½" high and 5½" long. On the other hand, if the streamline approaches the blade at an oblique angle of $\sigma = 22°$, as shown on the right by the green arrow, the "bump" is effectively shallower. It has the same absolute height, of 1½", but the effective length over which the bump extends is greater. It is now 5.93". The bump still has a circular shape, though. The effective length arising from the oblique approach was computed using the following trigonometric relationship:

$$\frac{5\frac{1}{2}"}{\cos 22°} = 5.93" \quad (9)$$

If the blade is infinitely long and does not rotate, a case can be made that the streamline angle σ is constant all along the airfoil. A practical yuloh is not infinite in length so there are "end-effects" which complicate the flow. Furthermore, a yuloh rotates, suggesting that the streamlines are not even straight, but curve when viewed from above as the water passes over the blade. That everything changes with time makes the situation even more complicated.

I will set all of these complexities aside and assume, for this preliminary analysis, that the water flows in a steady-state over the following profile, which is assumed to be infinitely long.



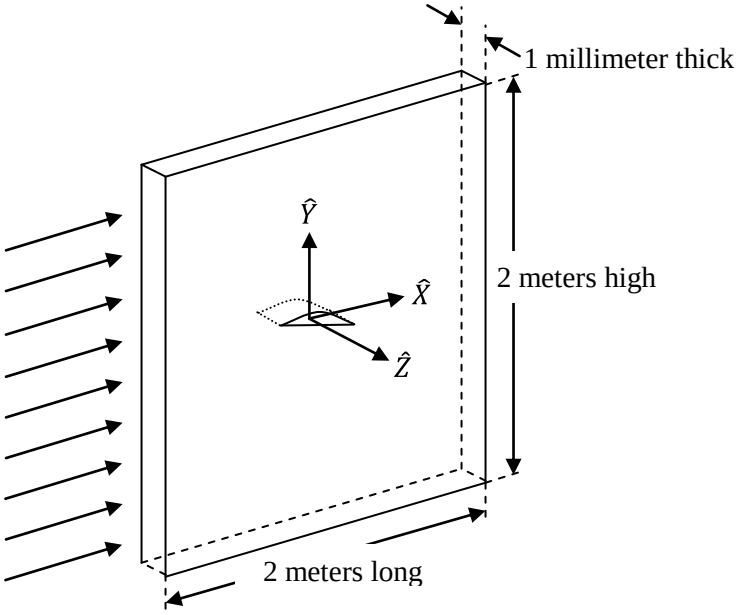
Based on the values set out in the above tables, I will look at angles of attack in the range from zero degrees to 30°, perhaps at five-degree increments. The relative speeds in the table above range from 3.36 feet per second to 5.84 feet per second. It might be useful to select three speeds -- 2 fps, 4 fps and 6 fps -- and focus on those three. This will provide 7 × 3 = 21 different cases, more than enough for now. Note that the angles of attack and relative speeds are not entirely uncorrelated from each other. One of the conclusions which can be drawn from the tables is that the angles of attack and relative speed are both

higher at the tip. To some degree, higher/lower angles of attack and higher/lower relative speeds go hand-in-hand.

For the next stage of the analysis, we are going to need, among other things, a formula from which we can draw the upper surface of the profile. It is a symmetrical segment of a circle, whose central height and length of base are known. One can use some geometry to figure out the equation of the circle which can generate the top surface. I have set out the details in Appendix "A". The equation of the generating circle for the profile just shown is the following:

$$\left.\begin{array}{ll} \text{Radius:} & R = \dfrac{5.93^2}{8 \times 1.5} + \dfrac{1.5}{2} = 3.68" \\ \text{Equation:} & x^2 + [y - (1.5 - R)]^2 = R^2 \text{ for } -2.965" \leq x \leq 2.965" \end{array}\right\} \quad (10)$$

Here is what we are going to do. We are going to set up a virtual wind tunnel and place inside it a section of the airfoil. Because we are assuming that the blade is infinitely long, the waterflow over every section will be the same. We are free to select any particular length of the blade we want. I have chosen to use a virtual wind tunnel which is only one millimeter thick. The following figure shows the apparatus, but it is not to scale.



The virtual wind tunnel is two meters long (measured in the direction of the waterflow), two meters high and, as I have already said, one millimeter thick. The co-ordinate frame of reference for all our simulations will be the $\hat{X}$-$\hat{Y}$-$\hat{Z}$ frame shown. I have used capital letters to identify these axes so there will be no confusion with any of the axes we have used before. The $\hat{X}$-$\hat{Y}$-$\hat{Z}$ frame is positioned inside the wind tunnel with its origin at the very geometric center of the wind tunnel. It is oriented so that the $\hat{X}$-axis points in exactly the same direction as the water is flowing. (As always, use of a wind tunnel assumes that the object is held still and the fluid moves with respect to the object, rather than the other way around.) The $\hat{Z}$-axis points straight out of the left side of the wind tunnel, when looking upstream.

The thin piece of the airfoil will be placed in the center of the wind tunnel. We will place it so the origin of the co-ordinate frame lies at the center of the flat bottom. We will also place it at the particular angle of attack we want to test.

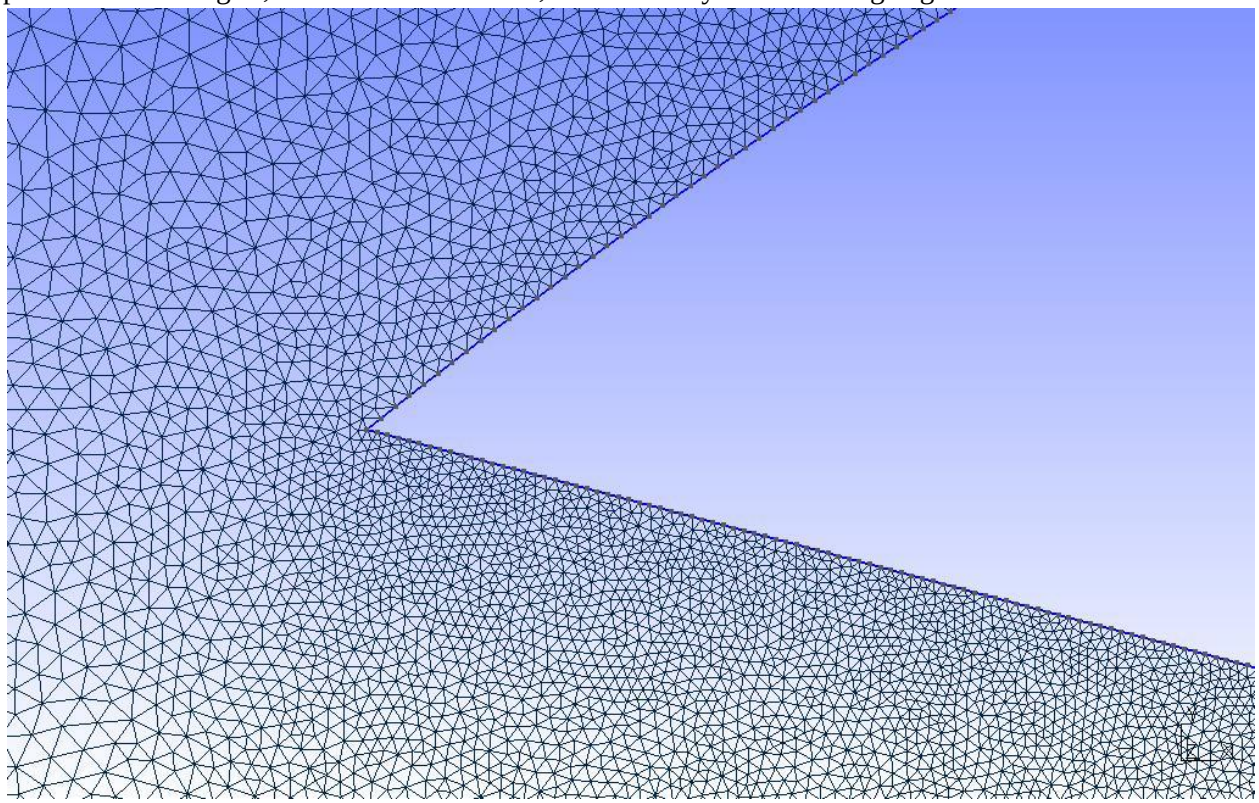
We are going to use computational fluid dynamics ("CFD") to simulate the flow of water around the airfoil. Use of CFD requires that the fluid inside the wind tunnel be divided up into a huge number of small bits. Its in calculations, CFD assumes that the conditions of the fluid are the same throughout each small bit and are affected by the conditions in the neighbouring bits. The more finely the fluid is divided up, the smaller the individual bits and the more detail can be extracted from the calculations. On the other hand, the more finely the fluid is divided up, the more small bits there are. One can be surprised at how quickly computer memory and processing speed are used up as the fluid is subdivided more and more finely.

Fortunately, the fluid does not need to be divided up into an infinite number of small bits in order to get very realistic results. Even better, there are ways to determine whether the number of bits being used is suitable for the dynamics being simulated.

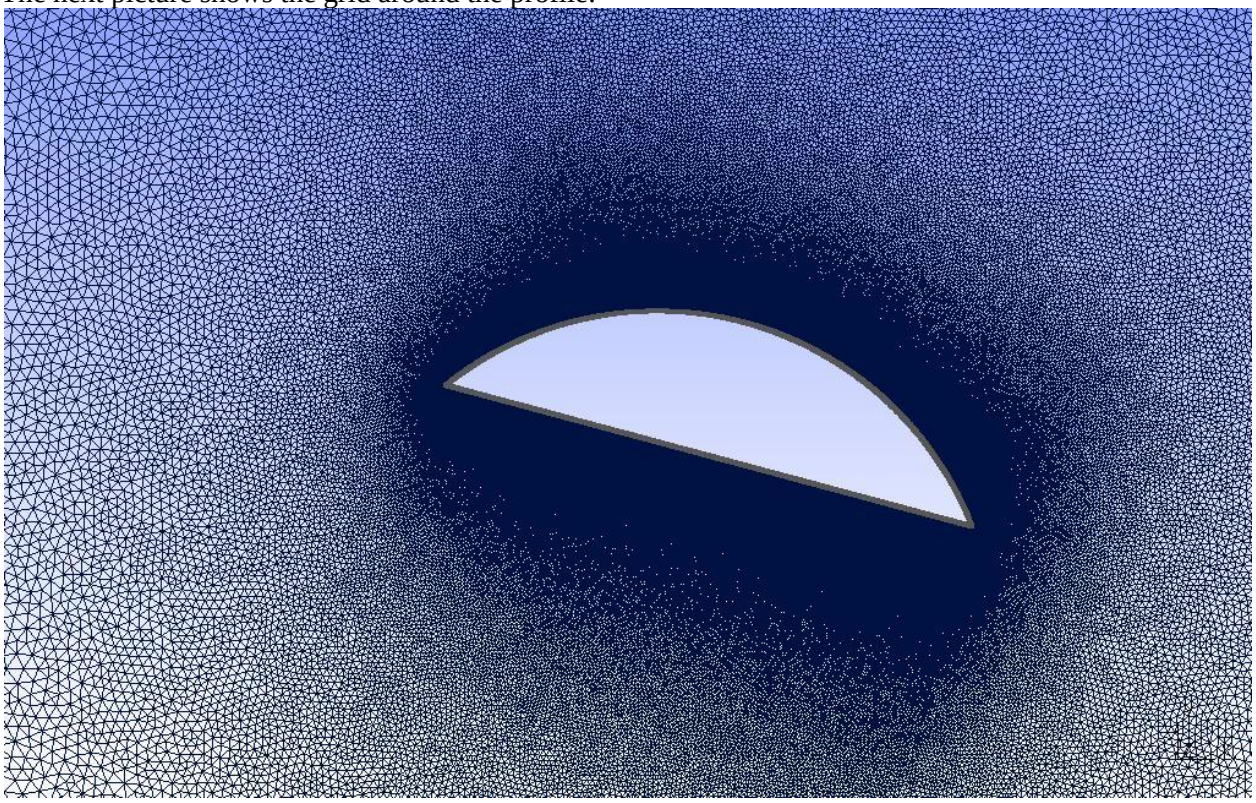
For the study at hand, I divided up the fluid using the following guidelines. The top and bottom surfaces of the profile were each divided into 1,000 short segments, of equal length along the reference chord. Since the arc length of the top surface is about six inches, the resulting segments have lengths of 0.006 inches, or about 0.15 millimeters. These segments will be the bases of the little triangles into which the cross-section of the fluid in the wind tunnel is divided, at least along the surface of the airfoil.

The two-meter length of each side of the main perimeter of the wind tunnel was divided into 80 segments, each of equal length, being 25 millimeters, or about one inch. These segments will be the sides of the triangles in the grid which border the edge of the wind tunnel.

Because we are assuming that the waterflow is the same at each section along the long axis of the blade, we will be simulating the flow in only two dimensions, looking at its pattern in a typical $\hat{X}$-$\hat{Y}$ plane. It is not necessary that we divide the fluid along the $\hat{Z}$-axis. In fact, the three-dimensional fluid in the wind tunnel is going to be divided up into little triangular prisms, whose cross-sections in the $\hat{X}$-$\hat{Y}$ plane are the triangles I have just described. The following figures are pictures of the grid I used for the case when the angle of attack is set to 15°. Note that the same grid can be used for any waterflow speed. The first picture shows the grid, also called the "mesh", in the vicinity of the leading edge.



The next picture shows the grid around the profile.



A picture showing the mesh across the whole wind tunnel is not informative. The size of the triangles, which are the triangular prisms seen in cross-section in these two pictures, is just too small for them to be resolved on a piece of paper. This mesh has about 550,000 triangles, which is not very large as these things go. Often, meshes require ten million or more elements.

The grid was constructed using a meshing program called "GMesh". GMesh is available on the internet as a free download. GMesh takes a text file prepared by the user and produces a three-dimensional mesh. The text file describes the geometry of the situation. For the yuloh, I prepared the text file with the help of a Visual Basic routine. For the sake of completeness, I have attached hereto as Appendix "D" a copy of the Visual Basic program which writes the text file GMesh uses as its data.

Let's move on. We need to consider the properties of the fluid. There are differences between sea water, lake water, river water, and so on. These differences are less significant than many of the other factors for which we have made assumptions or will be making assumptions. Therefore, I have used standard values quoted for the properties of water, which likely means fresh, pure water.

The two most important properties are the density and the viscosity. The density of water is 1,000 kg/m^3. The density is a nice round number. There is a historical reason for this. When early physicists began to quantify the relationship between physical volumes and mass, the substance they chose to use as a standard happened to be water. The symbol ρ, which is the Greek letter "rho", is usually used for density.

The most commonly used measure of viscosity is the "dynamic viscosity", usually represented by the symbol μ, the Greek letter "mu". The following table compares the dynamic viscosity of motor oil, water and air, all at room temperature.

Fluid	Dynamic viscosity, Ns/m^2
Motor oil	0.250
Water	0.001002
Air	0.0000173

Water is about 60 times more viscous than air. Motor oil is about 250 times more viscous than water.

In fluid dynamics, a variation of viscosity called the "kinematic viscosity" is frequently seen. It is defined as the dynamic viscosity divided by the density. It is usually represented by the symbol ν, the Greek letter "nu". Since the density of water is 1,000, the kinematic viscosity of water is one-thousandth of its dynamic viscosity. (Readers should note that, from this point on in this paper, I will be using S.I. units rather than English units.)

Viscosity is highly dependent on temperature. The following tables sets out the viscosity of water at 5°C and 20°C, which bound the range of water temperatures in which yulohs will likely be employed.

Temperature	Dynamic viscosity μ	Kinematic viscosity ν
5°C	$0.001519\ \mathrm{Ns/m^2}$	$1.519 \times 10^{-6}\ \mathrm{m^2/s}$
20°C	$0.001002\ \mathrm{Ns/m^2}$	$1.004 \times 10^{-6}\ \mathrm{m^2/s}$

Note that the viscosity varies by 50% over this quite narrow range of temperatures. Selecting the water temperature is probably more important than specifying whether the water is sea water or fresh water.

Let's talk for a minute about the type of fluid flow we can expect to find around the yuloh. Fluid flows around similar objects are frequently compared using their Reynolds numbers, named after the man who discovered the usefulness of said number. Airfoils are usually compared using a Reynolds number defined as:

$$Re = \frac{Vc}{\nu} = \frac{\rho V c}{\mu} \quad (11)$$

where V is the speed of the fluid, ν is the kinematic viscosity of the fluid and c is the length of the chord of the airfoil. (The chord is the length of the line segment connecting the leading edge to the tailing edge.) Before substituting values, let me say that the Reynolds number is the quotient obtained by dividing a measure of the inertial forces exerted on an airfoil by a measure of the viscous forces exerted on the airfoil. Inertial forces are those which affect the airfoil's momentum. Viscous forces are those which arise from the friction between the airfoil and the fluid.

Let's consider a small private aircraft cruising at 140 (statute) miles per hour, which is equivalent to 62.6 meters per second. It's wing has a chord length of six feet, or 1.83 meters. It is flying near sea level through air which has a dynamic viscosity of $1.73 \times 10^{-5}\ \mathrm{Nm/s^2}$ and a density of $1.225\ \mathrm{kg/m^3}$. The Reynolds number in this flight condition is:

$$Re = \frac{1.225\ \mathrm{kg/m^3} \times 62.6\ \mathrm{m/s} \times 1.83\ \mathrm{m}}{1.73 \times 10^{-5}\ \mathrm{Ns/m^2}} = 8{,}100{,}000 \quad (12)$$

As luck would have it, all of the dimensional units cancel each other out. Actually, luck has nothing to do with it. This was Reynolds great insight -- that dividing an inertial force by a viscous force would leave a dimensionless number.

The Reynolds number of a 747's airfoil is larger than this, but not by as much as you might think. The 747 flies faster and has a wider wing. Both of these increases appear in the numerator of the defined expression and cause the Reynolds number to increase. But, 747s cruise at a high altitude where the density of the air is quite a bit lower. This decrease also appears in the numerator, and tends to offset the other increases. The Reynolds number of a 747 is probably $10{,}000{,}000$.

Now, let's turn to the yuloh. The mid-point of the range of speeds we will look at is four feet per second, equivalent to 1.22 meters per second. The chord length is 5.93 inches, or 0.151 meters. We will assume the water temperature is 12.5°C and use the mid-point value from the table for the kinematic viscosity, $1.26 \times 10^{-6}\ \mathrm{m}^2/\mathrm{s}$. We can then compute the Reynolds number as:

$$Re = \frac{1.22\ \mathrm{m/s} \times 0.151\ \mathrm{m}}{1.26 \times 10^{-6}\ \mathrm{m}^2/\mathrm{s}} = 150{,}000 \quad (13)$$

This number is significantly smaller than that of the light aircraft -- less than two percent as much. The smaller Reynolds number merits an interpretation. The Reynolds number of the yuloh is smaller because the viscous forces (in the denominator) are relatively more important, and thus bigger, than the inertial forces (in the numerator). The effects of viscosity are about 50 times more important in the dynamics of a yuloh than they are in the dynamics of a light aircraft. The greater importance of the viscosity will manifest itself in such things as greater surface friction, more and larger vortices and so on. Not for nothing do canoers enjoy peeling vortices off the sides of their paddles. And Yulohers, too.

Because of the importance of viscous effects, we are going to have to use a CFD model which takes them into account. Readers should understand from this statement that users can choose CFD models which include some effects and exclude others. The Navier-Stokes equations are the bedrock of fluid dynamics. They are a comprehensive set of partial differential equations which account for the effects of space, time and viscosity. There are other formulations, too, but the one presented by Messrs. Navier and Stokes remains the most-commonly used one. The equations are virtually impossible to apply in closed form to any but the simplest problems. The partial differential equations can be discretized, but a host of numerical frustrations often make themselves known. Computers are not yet big or fast enough to permit one to grasp the holy grail, known as Direct Numerical Simulation ("DNS"), in which the size of elements in the mesh are reduced almost to the molecular level. Current computer capacity still requires the user to give up some aspect of the problem, such as changes taking place through time, in exchange for a sufficiently accurate representation of other aspects, such as the spatial dependence of a flow.

I have already given up two aspects of the problem, time and the third dimension. We are going to model the water flow as if was steady and does not change with time. By assuming that the blade is infinitely long, we also limit our attention to the two spatial dimensions in which the more interesting flow occurs. The component of the water flow along the long axis of the blade will be ignored.

The CFD model we use must include viscosity. For the simulations of the yuloh, I chose to use the Spalart-Allmaras model, named for the two investigators who pioneered its use. It is said to be good for two-dimensional flows like that we are looking at. Their model uses a parameter $\tilde{\nu}$ which is stored in the variable `nuTilda`. It is related to the kinematic viscosity ν but is not exactly equal to the kinematic viscosity. The independent variables in a Spalart-Allmaras model are `nuTilda`, the pressure and the two

components of the fluid's velocity vector. OpenFoam always simulates three dimensional velocities, but the third component can be zero, and will be zero when the geometry of the problem is so set.

The only other important parameters required to use OpenFoam are the specifications of the types of boundaries which surround the fluid. Although it is obvious, one sometimes loses sight of the fact that what is being simulated is the fluid in the wind tunnel, not the object. To be precise, the fluid in our wind tunnel is not simply a rectangular parallelepiped surrounded by six sides. The parallelepiped has a hole in it, from the left side to the right side, in the shape of the blade's profile. The fluid is bounded by 2,006 plane surfaces, where the extra 2,000 surfaces consist of 1,000 rectangles which define the top surface and 1,000 rectangles which define the bottom surfaces. Both the top and bottom surfaces have been discretized into 1,000 short line segments, which approximate the shape of the profile. When it is extruded from one side of the wind tunnel to the other, each such line segment sweeps out a small rectangle.

Specifying the types of these 2,006 boundaries is important. The right and left walls of the wind tunnel have the type "empty". This means that the CFD routine does not do any calculations on these two surfaces. That is what we want when we want the third dimension of the geometry -- the $\hat{Z}$-axis -- to be infinite and unchanging in both directions.

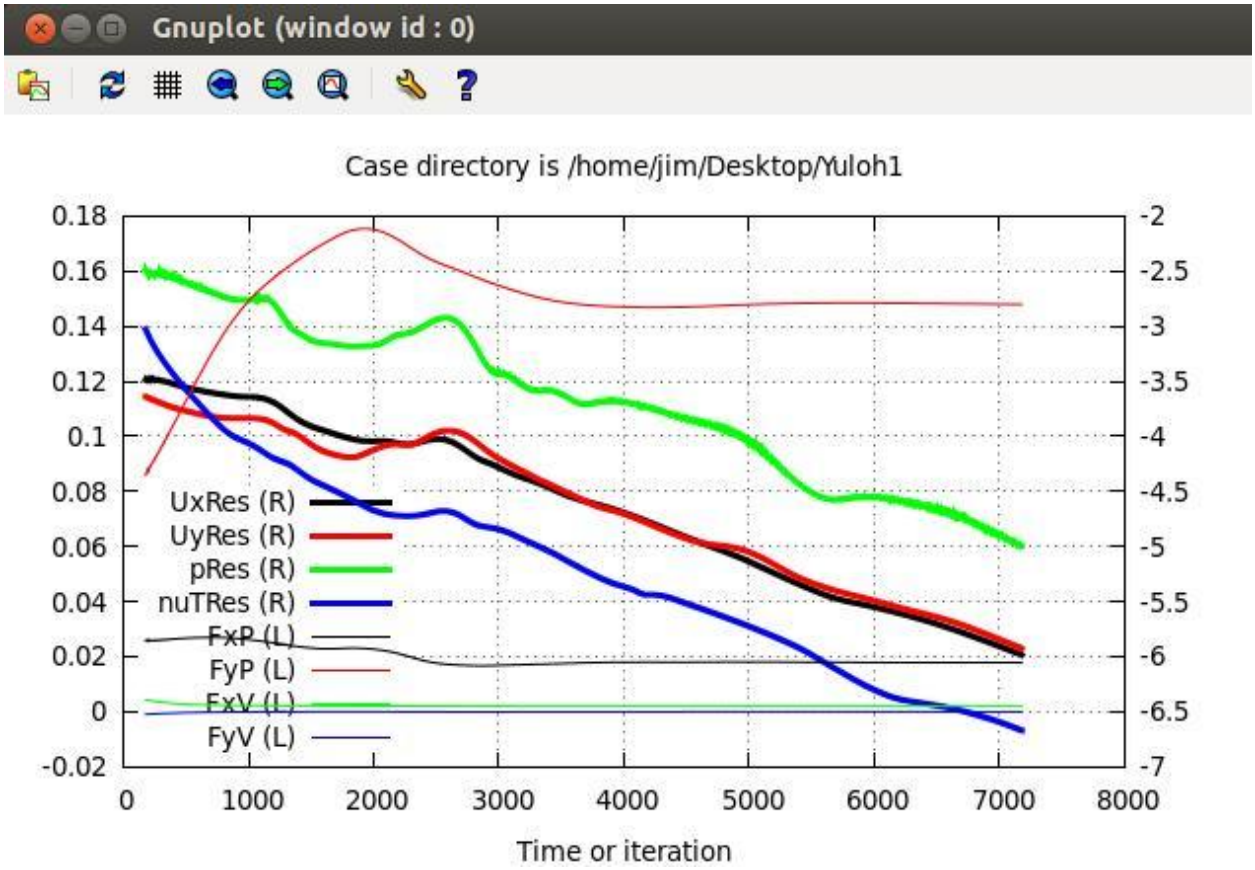
The upstream and downstream faces of the wind tunnel have the type "patch". They are faces on which we have to specify some properties of the fluid that passes through them. The upstream face is called the "inlet". We will specify the fluid's velocity everywhere on this face. In our problem, the velocity will be uniform across this face, at four feet per second or six feet per second, or whatever speed we select. Of course, all quantities are expressed in S.I. units, not English units. The downstream face of the wind tunnel is called the "outlet". We will specify the fluid's pressure everywhere on this face. In our problem, the pressure will be uniform across this face. Interestingly, we call set an arbitrary value for the pressure across this face. For convenience, we will set the pressure here to zero. Let me explain why the numerical value we select does not matter and need not be the absolute pressure one would measure there using a manometer. The net force which the fluid exerts on the airfoil arises from differences in the pressure on one side relative to that on the other. It is the difference in pressure from point to point along the surface, and not the absolute pressure, which determines the force. Adding or subtracting a constant pressure everywhere throughout the fluid does not change the differences and therefore does not change the computation of the physical force in which we are interested. Notionally, we are simply subtracting a fixed amount of pressure (in an unknown amount) from the pressure everywhere in the fluid so that the pressure across the downstream outlet has the numerical value zero. The numerical procedure will do what it has to do to ensure that the oncoming speed of the water is uniform across the inlet and that the pressure is uniform across the outlet.

The top and bottom of the wind tunnel have the type "symmetryPlane". A symmetry plane is one across which the properties of the fluid do not change. If the pressure has some value at a point just below the top face of the wind tunnel, it will have the same value just above the wind tunnel at that same $\hat{X}$-$\hat{Z}$ co-ordinate. If the velocity of the fluid has some value at a point just above the bottom face of the wind tunnel, it will have the same value just below the wind tunnel at that same $\hat{X}$-$\hat{Z}$ co-ordinate. That the face is a symmetry plane does not mean that the properties of the fluid must be the same everywhere on the face, just that they are pointwise the same across the face. This is a sufficient boundary constraint for our analysis even though it is less restrictive than a condition that would specifically set the velocity and pressure everywhere on the face. The latter specification would succeed, but it is easier for the calculations to have some flexibility.

Lastly, we have to deal with the 2,000 rectangles which define the surface of the airfoil. They have the type "wall". As the name suggests, they are physical walls over which the fluid moves. Or not. Indeed,

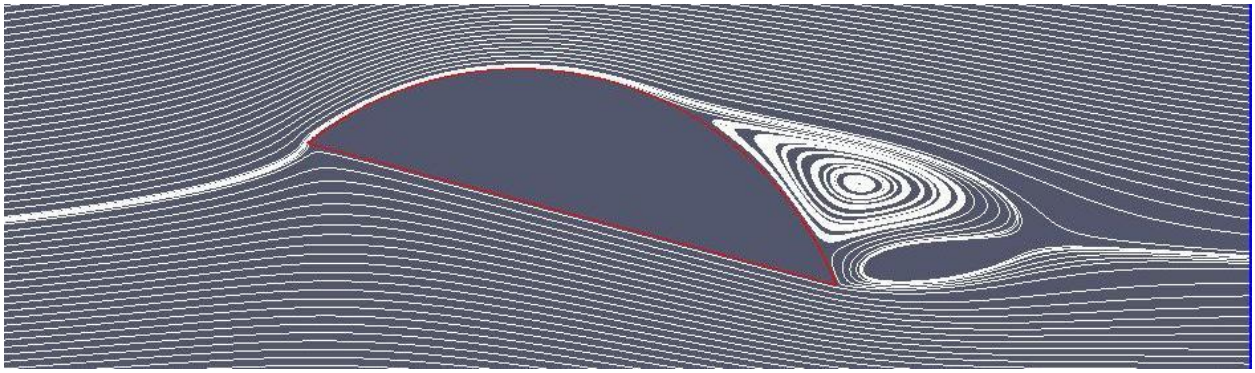
the principal feature we will specify for the walls is a "no slip" condition. The water cannot slide along the surface -- viscosity prevents that. Instead, a boundary layer will form. I have listed in Appendix "B" the ten text files which contain the information OpenFoam was given for the base case in this analysis. For certainty's sake, let me re-state that the base case consists of a blade carved from a stock piece of 2×6 lumber, at an angle of attack of 15° in water with a relative speed of four feet per second and a temperature of 12.5°C.

The following figure shows the progress of convergence during the simulation of the base case.



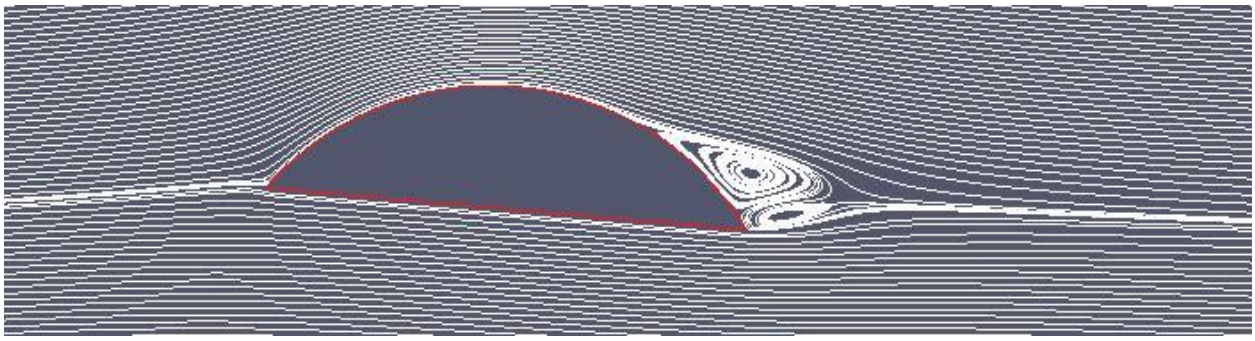
The procedure converged, almost monotonically, in about 7,200 iterations. By convergence, I mean that the simulation was ended when the fractional differences in all four independent variables were less than 1×10^{-5} from one iteration to the next.

The following figure is a picture of the streamlines around the airfoil in the base case.

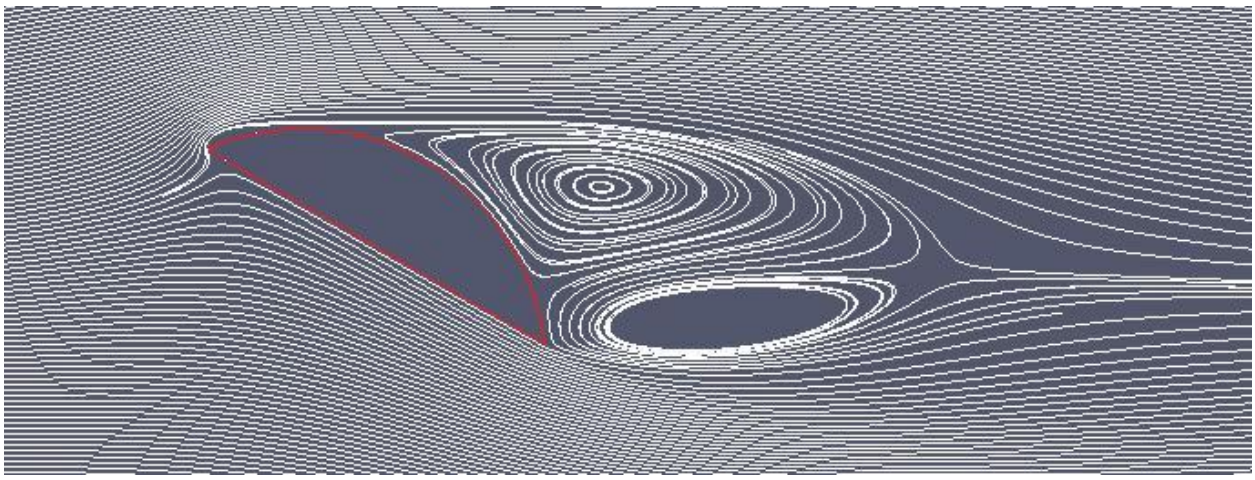


There is some serious separation of flow at about 60% of chord. The airfoil is quite blunt. It has a thickness ratio of 1.25"/5.93", or 21%.

The following picture shows the streamlines at an angle of attack of 5° and the same water speed: four feet per second. There is still separation, but it does not commence until about 75% of chord. Likely, redusing the thickness will alleviate some of the stall.



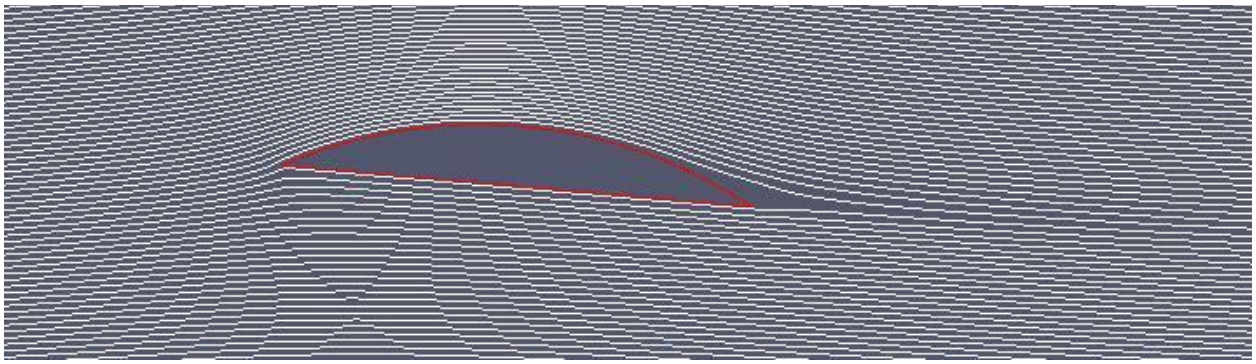
The following picture shows the streamlines at an angle of attack of 30° but a different water speed: six feet per second. This is intended to represent the conditions near the tip of the blade.



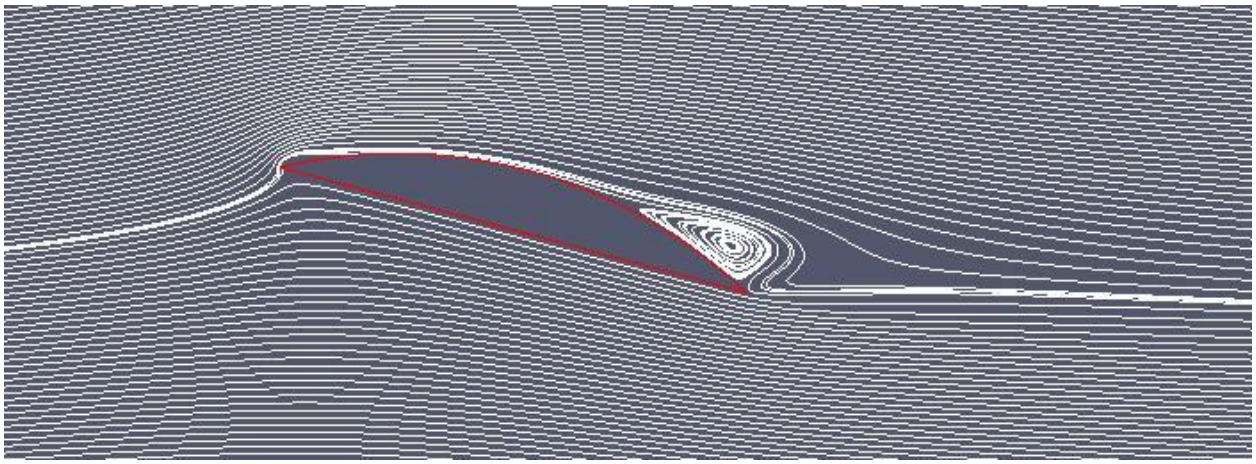
At 30°, the blade is not really functioning as an airfoil at all. The water separates from the top surface as soon as it is able to go straight downstream.

Conclusion: A 30° angle of attack is simply ineffective. The kinematic analysis we did above suggests that the angle of attack at the tip of the blade is about 10° greater than the angle of attack at the stock (the inboard end of the blade). We should probably aim for an angle of attack of 15° at the tip and 5° at the stock.

It is easy to see that the cross-section we have been using is too thick. As an alternative to carving the blade from a stock piece of 2 × 6 lumber, let's assume instead that we shape it from a stock piece of 1 × 6 lumber. The true dimensions of 1 × 6 lumber are ¾" × 5½". The following picture shows the streamlines of this thiner blade at a 5° angle of attack.



This is much more satisfactory. (However, there is more to determining the success of an airfoil than looking at streamtracers. We have not yet looked at the quanta of the forces, which is a vital ingredient. Even so, it is clear from this picture that this blade is more efficient than the thicker ones we looked at above.) The following picture shows the streamlines at a 15° angle of attack. Even though this is intended to represent the tip of the blade, I still used a water speed of four feet per second in the simulation for the next picture.



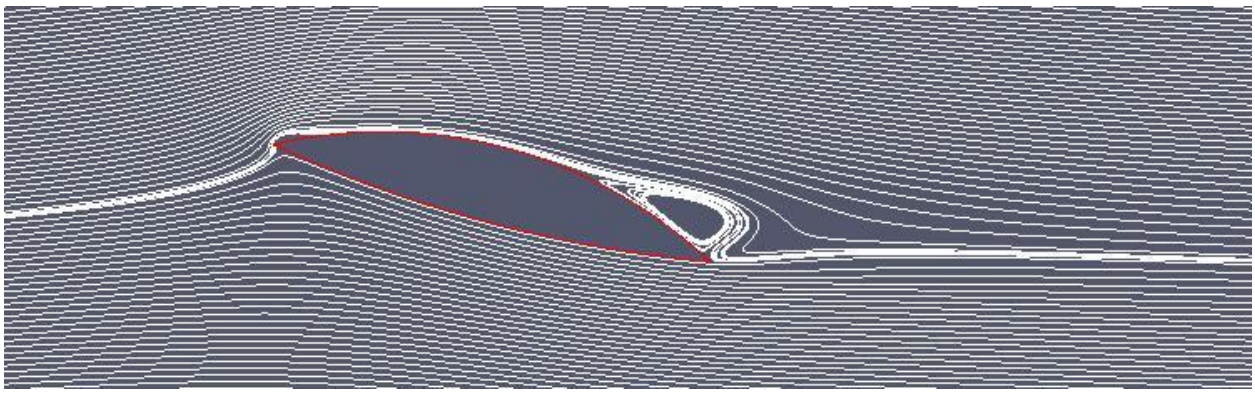
There is still separation. Whether it is "too much" I cannot say. It is, however, much better than the patterns at 15° and 30° using the thicker airfoil.

Incidentally, the simulation of this thinner 1 × 6 blade at a 30° angle of attack did not converge. I terminated the simulation after about 30,000 iterations. The residuals oscillated with a period of about 5,000 iterations. Such oscillations of the residuals are a good indication that the flow is unstable, and that the procedure is having trouble determining whether the flow is separated or not. A quick look at the pattern of streamlines which existed when I terminated the simulation shows that the flow separated at the leading edge. The curved top surface of the blade was not exerting any control over the airflow. It does not matter whether this simulation would ultimately have converged or not -- it was plain that the blade would not be useful at a 30° angle of attack.

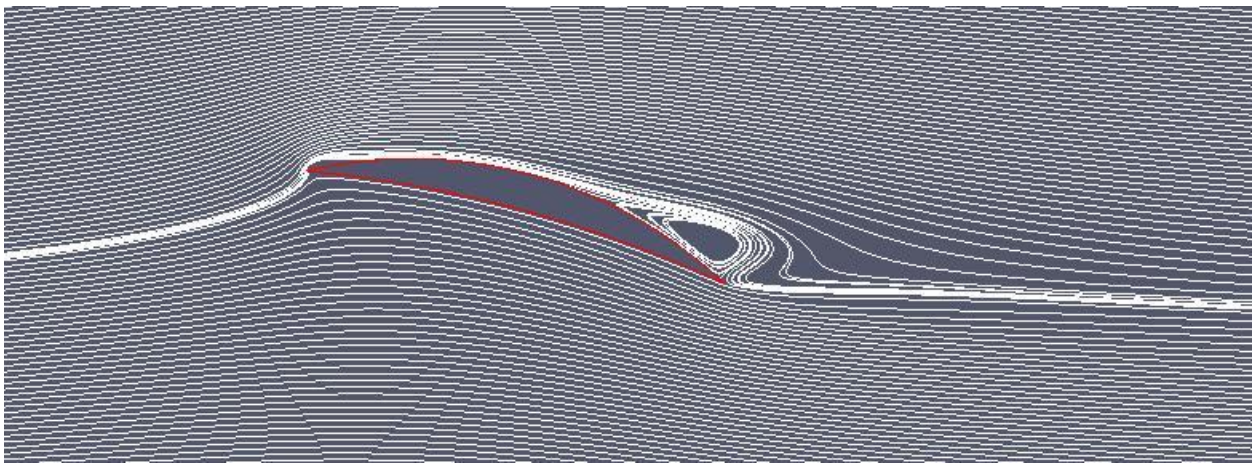
I next wanted to explore whether changes to the bottom of the blade, which up until this point has been the flat rear side, might help. Since the Visual Basic program which writes the text file for GMesh already includes the necessary code to draw a circular profile, I decided to give the bottom surface of the blade a circular profile as well. I considered two alternatives. One alternative had the bottom surface protruding from the blade, in other words, being a convex surface. In the second, I created a trough in the bottom by making the circular surface concave. The following pictures show the shape of the bottom

surface as well as the pattern of streamlines. In both cases, I used a thickness of one-quarter inch for the bottom surface, which made it one-third of the thickness of the top surface. In both case shown now, the angle of attack is 15° and the water speed is four feet per second.

The following picture shows the streamlines around the blade with the convex bottom surface.



The following picture shows the streamlines around the blade with the concave bottom surface.



It is not immediately clear from the pictures which of the convex or concave surfaces is better, or whether either is better than the flat surface. The following analysis suggests that the concave surface is superior.

Now, it is time to look at the magnitude of the forces. OpenFoam returns two types of forces: pressure-induced forces and viscosity-induced forces. The former arise from the static pressure exerted by the water on the surfaces. The latter arise from friction as water is forced to slide along the surfaces. Both sets of forces depend only on the water conditions on the very surface of the blade. What the water does even a small distance away from the surface is irrelevant except insofar as it affects what happens on the actual surface itself. The total force on the airfoil is the vector sum of the two types of forces. OpenFoam reports the magnitudes in all three spatial dimensions. In our case, since we are doing a two-dimensional study only, the forces calculated by OpenFoam in the $\hat{Z}$-direction, which is along the long axis of the yuloh, are zero.

I have set out in Appendix "C" attached the detailed results. Since OpenFoam does its calculations in S.I. units, it reports forces in units of Newtons. A Newton of force has about the same "heft" as the weight of the paddy in a McDonald's quarter-pounder. The results I will describe in the next few paragraphs are in pounds (of force), each one of which is 4.45 Newtons.

To give some idea of what we are talking about, the following table sets out the magnitudes of the forces in the base case.

Blade	Speed	Angle of attack	Pressure forces		Viscous forces		Total forces		L/D ratio
			$\hat{X}$	$\hat{Y}$	$\hat{X}$	$\hat{Y}$	$\hat{X}$	$\hat{Y}$	
2×6	4 fps	15°	4.0	33.2	0.2	0	4.4	33.2	7.5

The components of force reported in the table are per meter of span-wise length of the blade. Remember that we set up the wind tunnel so that is was only one millimeter thick. If we multiply the forces computed by OpenFoam by a factor of 1,000, we get the forces which would be exerted on a one-meter long section of the blade. The viscous forces are quite small compared with the pressure forces. Although the sliding friction is not that large, it has a vital influence on the pattern of the flow, which in turn determines the static pressure.

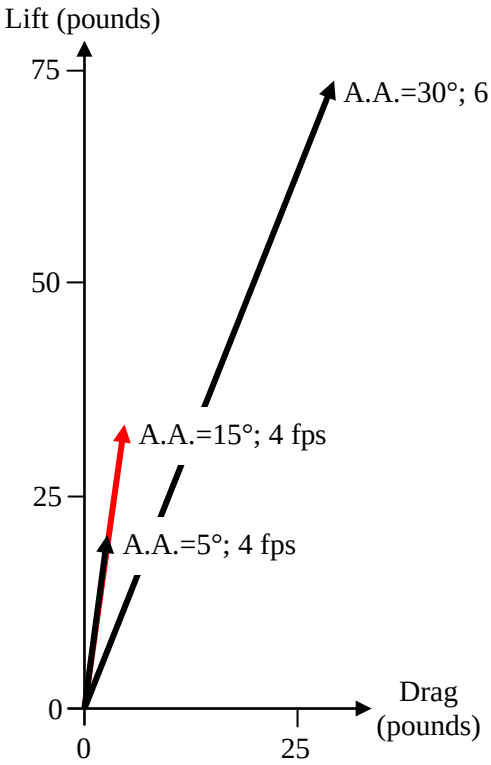
OpenFoam reports components in the $\hat{X}$- and $\hat{Y}$-axes, which are not the same as the axes we used in the analyses above. It is enough for our purposes to know that the $\hat{X}$-axis is the one which points straight in the direction in which the unaffected water far upstream is flowing. The $\hat{Y}$-direction is perpendicular to this, pointing towards the curved top surface.

Let me flog this point. It is important to bear in mind that the two OpenFoam axes are not the same as the $\hat{x}_6$ and $\hat{y}_6$ axes which are fixed to the yuloh's blade. Roughly speaking, though, the $\hat{X}$-direction forces are the ones which retard the motion of the blade through the water. They represent the drag which the blade must overcome, or more precisely which the operator must overcome, to cause the blade to move through the water. We assumed above that the blade is five feet long. That is equivalent to about 1.52 meters. If the drag force is 4.4 pounds per meter length (from the table), then the drag on the blade would be 1.52 times that, or 6.7 pounds. If the effective distance from the fulcrum to the center of pressure of the blade is three times greater than the effective distance from the fulcrum to the center-of-effort of the operator, then the operator will need to exert $3 \times 6.7 = 20.1$ pounds of force on the loom to cause the blade to move.

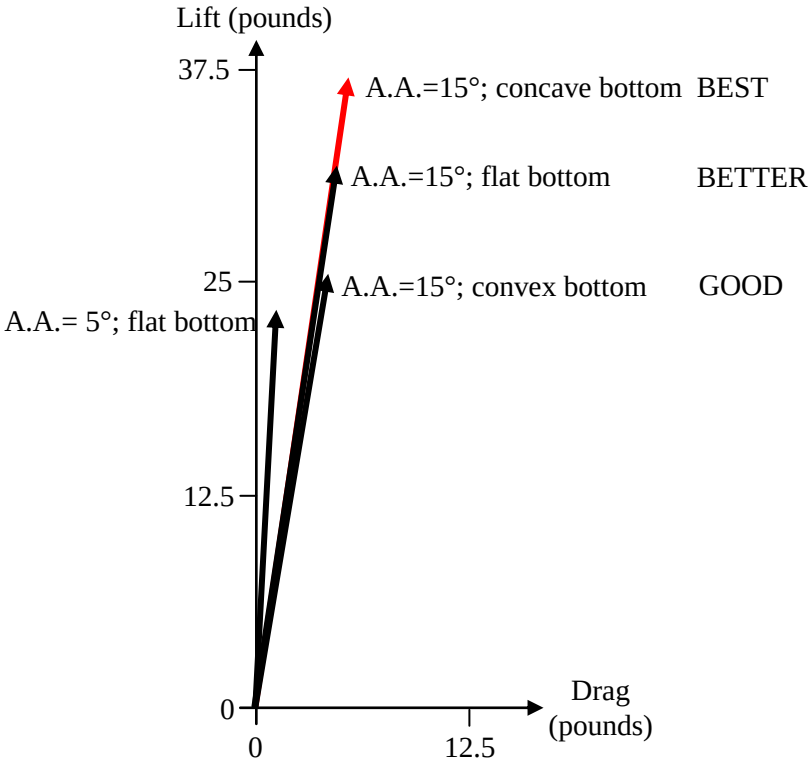
Now, look at the lift force, 33.2 pounds. The lift force is 7.5 times greater than the drag force. This is the mystery and beauty of an airfoil. The blade generates lift many multiples greater than the force required to move it through the water. True, this lift is not all acting in the direction we would wish. Ideally, all of the lift would act in the direction of the boat's path. Since the yuloh is angled downwards, a significant part of the lift acts downwards instead of ahead. We assumed above that the slant angle was 40°. At that angle, only 64% of the lift acts forwards. (Aside: $\sin 40° = 0.643$). Furthermore, the yuloh sweeps from side-to-side, so a component of the lift is directed off-course at all times except when the yuloh is amidships. Fortunately, the sweep angles are not that big -- we assumed a maximum of 12.4° -- so the component of lift lost in yawing the boat is not that large.

The ideal yuloh would have a high lift-to-drag ratio ("L/D"), which would mean it is efficient, but it also needs to have forces with suitable magnitudes. The operator has to overcome the drag force, which cannot be made unmanageably large. Nor can it be made too small. As a yuloh designer, we have pretty good control over the magnitude of the forces. If we need to increase the forces, we can always make the yuloh longer or, perhaps, make the blade wider. The blade cannot be made too long, though. The longer we make the blade the greater the difference between the angles of attack at the tip and stock. The expected depth of the waterway may also impose a practical limit on the length of the yuloh.

The following chart compares the total lift and total drag forces for the three cases in which the 2 × 6 blade was tested.



The steepness of the lines is proportional to the lift-to-drag ratio. A steeper line is better. As is usually the case, the magnitudes of the forces generally increase as the angle of attack increases. (Note that a contributor to the greater forces in the 30° case comes from the higher water speed, six feet per second rather than only four feet per second.) Even though the flow at the 30° angle of attack results in much bigger forces, it is much less efficient. The lift force is only about 2.5 times the drag force. Although I derided this pattern of flow, and pointed out its early separation, it nevertheless produces a lot more lift than do the lower angles of attack. The following chart shows the lift and drag forces which the 1 × 6 blade generates.

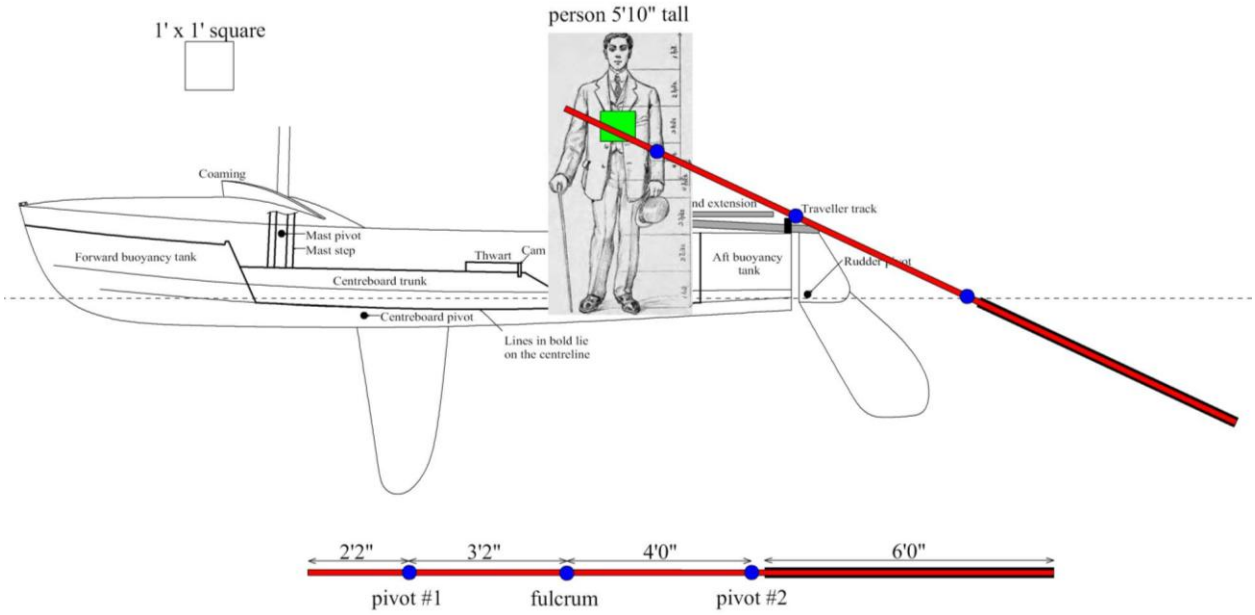


The airfoil is more efficient at an angle of attack of 5° than at 15°. If we could, it would be nice just to arrange things so the angle of attack would be 5° all along the blade. But, we cannot. The geometry of the situation is such that the angle at the tip is going to be near 15° whether we like it or not. It looks like the conditions at the tip are going to be the determining factor in selecting a profile.

The flat bottom, convex botton and concave bottom all have about the same efficiency. But the magnitude of the lift generated by the concave profile is clearly the best.

Where do we go from here?

I am interested in the yuloh as propulsion for a Wayfarer and intend to make a test yuloh. The following figure is a fairly accurate side view of the Wayfarer. In the cockpit stands a 5'10" man, who happens to be the same height as me.



To be conservative, I have left the man standing on the bottom of the boat. Anythng which would raise him up, like floor boards, would help to steepen the yuloh, but I do not want to consider such enhancements at this early stage.

I have marked with a green rectangle the spot on the man's torso where the loom should cross him. The fulcrum, marked with one of the blue dots, will be mounted on the transom so the yuloh just clears the traveler rack. These considertions establish the slope the yuloh must take. The yuloh will clear the rudder by a good margin. It will be handy to be able to deploy the yuloh whilst leaving the rudder in its pintles. These considerations, plus the desired length of the blade under water, also establish the total length of the yuloh.

The horizontal object in red at the bottom of the figure is the yuloh lying flat. I am going to make the blade six feet long, notwithstanding that the analysis was based on a five foot length. If necessary, it is easier to cut a section off than to extend the blade. In addition to the fulcrum, I have shown two other blue dots. In my test yuloh, I intend these to be single-axis hinges. I would like to experiment with different angles. Adjustable hinges at the two points shown will permit various angles of droop and anti-droop to be tested against various blade slant angles. The hinges will permit the sections to be adjusted in the vertical plane, but not the horizontal plane. The hinge labeled pivot #1 will be immediately at the

sculler's aft hand. The hinge labeled pivot #2 will be as close as practical to the top of the blade, to minimize the size of the cone swept out when the blade's twist angle is reversed at stroke ends.

January 2014
Jim Hawley

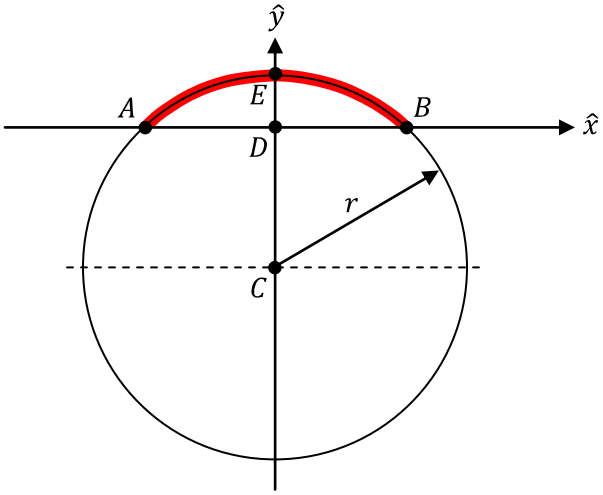
P.S. A couple of weeks in front of the computer screen, followed by a couple of days in the shop and a couple of hours of testing, is no substitute for 4,000 years of Darwinian selection on the Yangtze River.

P.P.S. The aborigines of Australia may have mastered aerodynamics with their boomerangs, but the Chinese mastered hydrodynamics with their yulohs.

Appendix "A"

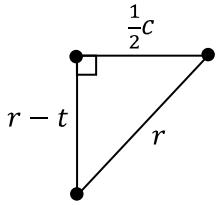
Equations of an airfoil comprised of opposing circular arcs

Let's consider an airfoil which is symmetrical, and whose upper and lower surfaces are arcs of circles.



The figure here shows the circle which generates the upper surface highlighted in red. Points A and B are the leading and trailing edges. Which one is which makes no difference since the airfoil is symmetrical about its midpoint. The circle which generates the upper surface of the profile has its center at point C. Our goal is to develop the shape which: (i) has a specified chord length (distance $\overline{AB}$), for which I will use the symbol c and (ii) has a specified thickness (distance $\overline{DE}$), for which I will use the symbol d. All we need to calculate is the radius r of the generating circle.

Once we have found the correct answer, then triangle CDB will be a right triangle whose three sides have the lengths shown here. We can invoke the Pythagorean Theorem to write:



$$r^2 = \left(\frac{1}{2}c\right)^2 + (r-t)^2 \qquad (A1)$$

We can solve this for radius r using the following steps:

$$\begin{aligned} & r^2 = \left(\frac{1}{2}c\right)^2 + (r-t)^2 \\ \rightarrow \quad & r^2 = \frac{1}{4}c^2 + r^2 - 2rt + t^2 \\ \rightarrow \quad & 2rt = \frac{1}{4}c^2 + t^2 \\ \rightarrow \quad & r = \frac{c^2}{8t} + \frac{t}{2} \qquad (A2) \end{aligned}$$

Let's try an example. Suppose we want an airfoil with a chord length of six inches $(c = 6)$ and a thickness of two inches $(t = 2)$. The radius of the generating circle is four and one-quarter inches, computed as follows:

$$r = \frac{6^2}{8 \times 2} + \frac{2}{2} = 4.25 \text{ inches}$$

In Cartesian co-ordinates x and y, the equation of a circle with radius r whose center is located at point (x_0, y_0) is $(x - x_0)^2 + (y - y_0)^2 = r^2$. We can insert the center of our generating circle $(0, t - r)$ to write the equation of our generating circle as:

$$x^2 + [y - (t - r)]^2 = r^2 \qquad (A3)$$

Often, we will want to draw the shape of the upper surface. To do this, one normally selects several different points along the $\hat{x}$-axis and, for each x-value, calculates the corresponding value of y. This is done most efficiently if we re-arrange the equation for the circle as follows:

$$\begin{aligned} & x^2 + [y - (t - r)]^2 = r^2 \\ \rightarrow \quad & [y - (t - r)]^2 = r^2 - x^2 \\ \rightarrow \quad & y - (t - r) = \sqrt{r^2 - x^2} \\ \rightarrow \quad & y = \sqrt{r^2 - x^2} + t - r \qquad (A4) \end{aligned}$$

The same equations can be used to create a bottom surface for the profile. If the bottom is to be concave, so that it has the same upward curve as the top surface, then the radius of the generating circle will be larger than the radius of the generating circle for the top surface. Indeed, if the radius is made very large, for example 1×10^{20}, then the bottom surface will be flat. To construct a convex bottom surface, set the offset y-values in Equation (A4) negative, so they fall below the reference chord line between the leading and trailing edges.

Appendix "B"

Control files for the base case OpenFoam simulation run

This appendix contains a listing of the ten files used to control the OpenFoam simulation of the base case. The name of the "case directory" for the base case is `Yuloh_4fps_15deg_2By6Blade`. The 10 files are located in the following sub-directories:

```
Yuloh_4fps_15deg_2By6Blade/
 |
 |--0/
 |   |
 |   |--nut
 |   |--nuTilda
 |   |--p
 |   |--U
 |
 |--constant/
 |   |
 |   |--polyMesh/
 |   |   |
 |   |   |--boundary
 |   |
 |   |--RASProperties
 |   |--transportProperties
 |
 |--system
 |   |
 |   |--controlDict
 |   |--fvSchemes
 |   |--fvSolution
```

This directory structure does not name all of the files used or produced by an OpenFoam run, but the ten files listed below would be sufficient to recreate the results.

Listing of file `0/nut`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      nut;
}
// Set the freestream value of nut to one-tenth of nuTilda.
// If nuTilda = 0.001865, then nut = 0.0001865.
dimensions      [0 2 -1 0 0 0 0];
internalField   uniform 0.0001865;
boundaryField
```

```
{
    Inlet
    {
        type            freestream;
        freestreamValue uniform 0.0001865;
    }
    Outlet
    {
        type            freestream;
        freestreamValue uniform 0.0001865;
    }
    RightWall
    {
        type            empty;
    }
    LeftWall
    {
        type            empty;
    }
    Top
    {
        type            symmetryPlane;
    }
    Bottom
    {
        type            symmetryPlane;
    }
    "Segment.*"
    {
        type            nutUSpaldingWallFunction;
        value           uniform 0;
    }
}
```

Listing of file `0/nuTilda`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      nuTilda;
}

// Calculate nuTilda = sqrt(1.5) * UIl, where
//   U = 1.2192 m/s
//   I = 0.025 is the estimated turbulent intensity
//   l = 1 centimeter is the estimated length scale
// Then, nuTilda = 0.000373
// Set the freestream value of nuTilda to five times this.

dimensions      [0 2 -1 0 0 0 0];
internalField   uniform 0.001865;
boundaryField
```

```
{
    Inlet
    {
        type            freestream;
        freestreamValue uniform 0.001865;
    }
    Outlet
    {
        type            freestream;
        freestreamValue uniform 0.001865;
    }
    RightWall
    {
        type            empty;
    }
    LeftWall
    {
        type            empty;
    }
    Top
    {
        type            symmetryPlane;
    }
    Bottom
    {
        type            symmetryPlane;
    }
    "Segment.*"
    {
        type            fixedValue;
        value           uniform 0;
    }
}
```

Listing of file `0/p`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
dimensions      [0 2 -2 0 0 0 0];
internalField   uniform 0;
boundaryField
{
    Inlet
    {
        type        zeroGradient;
    }
    Outlet
    {
        type        fixedValue;
```

```
        value       uniform 0;
    }
    RightWall
    {
        type        empty;
    }
    LeftWall
    {
        type        empty;
    }
    Top
    {
        type        symmetryPlane;
    }
    Bottom
    {
        type        symmetryPlane;
    }
    "Segment.*"
    {
        type        zeroGradient;
    }
}
```

Listing of file 0/U

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}

// 2 feet per second = 0.06096 m/s
// 4 feet per second = 1.2192 m/s
// 6 feet per second = 1.8288 m/s

dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (1.2192 0 0);

boundaryField
{
    Inlet
    {
       type         fixedValue;
       value        uniform (1.2192 0 0);
    }
    Outlet
    {
        type        zeroGradient;
    }
    RightWall
    {
        type        empty;
```

```
    }
    LeftWall
    {
        type        empty;
    }
    Top
    {
        type        symmetryPlane;
    }
    Bottom
    {
        type        symmetryPlane;
    }
    "Segment.*"
    {
        type        fixedValue;
        value       uniform (0 0 0);
    }
}
```

Partial listing of file `constant/polyMesh/boundary`

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       polyBoundaryMesh;
    location    "constant/polyMesh";
    object      boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

2006
(
    RightWall
    {
        type            empty;
        nFaces          553962;
        startFace       829623;
    }
    LeftWall
    {
        type            empty;
        nFaces          553962;
        startFace       1383585;
    }
    Top
    {
        type            symmetryPlane;
        nFaces          160;
        startFace       1937547;
    }
    Outlet
    {
```

```
        type            patch;
        nFaces          160;
        startFace       1937707;
    }
    Bottom
    {
        type            symmetryPlane;
        nFaces          160;
        startFace       1937867;
    }
    Inlet
    {
        type            patch;
        nFaces          160;
        startFace       1938027;
    }
    Segment.1
    {
        type            wall;
        nFaces          1;
        startFace       1938187;
    }
    Segment.2
    {
        type            wall;
        nFaces          1;
        startFace       1938188;
    }
```

[Records for Segment.3 through Segment.1998 removed from listing.]

```
    Segment.1999
    {
        type            wall;
        nFaces          1;
        startFace       1940185;
    }
    Segment.2000
    {
        type            wall;
        nFaces          1;
        startFace       1940186;
    }
)
```

Listing of file `constant/RASProperties`

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      RASProperties;
```

```
}
RASModel        SpalartAllmaras;
turbulence      on;
printCoeffs     on;
```

Listing of file `constant/transportProperties`

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}

// Water
// Temperature   Density-----  Dynamic viscosity-  Kinematic visc---
//    5.0 deg    1,000 kg/m^3  1.519x10^-3 Ns/m^2  1.519x10^-6 m^2/s
//   12.5 deg    1,000 kg/m^3  1.261x10^-3 Ns/m^2  1.262x10^-6 m^2/s
//   20.0 deg    1,000 kg/m^3  1.002x10^-3 Ns/m^2  1.004x10^-6 m^2/s

transportModel  Newtonian;
nu              nu [0 2 -1 0 0 0 0] 1.262E-6;
rho             rho [ 1 -3 0 0 0 0 0 ] 1000;
```

Listing of file `system/controlDict`

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
application     simpleFoam;
startFrom       latestTime;
startTime       0;
stopAt          endTime;
endTime         100000;
deltaT          1;
writeControl    timeStep;
writeInterval   250;
purgeWrite      3;
writeFormat     ascii;
```

```
writePrecision  7;
writeCompression off;
timeFormat      general;
timePrecision   6;
runTimeModifiable true;
//
// Function to print forces exerted on a section of the blade.
//
functions
{
  ForceOnBlade
  {
    type                 forces;
    functionObjectLibs   ( "libforces.so" );
    patches              (
                          "Segment.*"
                         );
    rhoName              rhoInf;
    pName                p;
    UName                U;
    log                  true;
    rhoInf               1000;
    CofR                 ( 0 0 0 );
    outputControl        timeStep;
    outputInterval       1;
  }
};
```

Listing of file `system/fvSchemes`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}

ddtSchemes
{
    default         steadyState;
}

gradSchemes
{
    default         Gauss linear;
    grad(p)         Gauss linear;
    grad(U)         Gauss linear;
}

divSchemes
{
    default                     none;
```

```
    div(phi,U)                    Gauss linearUpwind grad(U);
    div(phi,nuTilda)              Gauss linearUpwind grad(nuTilda);
    div((nuEff*dev(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default                              none;
    laplacian(nuEff,U)                   Gauss linear corrected;
    laplacian((1|A(U)),p)                Gauss linear corrected;
    laplacian(DnuTildaEff,nuTilda)       Gauss linear corrected;
    laplacian(1,p)                       Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
    interpolate(U)  linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    p               ;
}
```

Listing of file `system/fvSolution`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
solvers
{
    p
    {
        solver          GAMG;
        tolerance       1e-10;
        relTol          0.05;
        smoother        GaussSeidel;
        nPreSweeps      0;
        nPostSweeps     2;
        cacheAgglomeration true;
        nCellsInCoarsestLevel 10;
        agglomerator    faceAreaPair;
```

```
        mergeLevels     1;
    }
    U
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        nSweeps         2;
        tolerance       1e-10;
        relTol          0.05;
    }
    nuTilda
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        nSweeps         2;
        tolerance       1e-10;
        relTol          0.05;
    }
}
SIMPLE
{
    nNonOrthogonalCorrectors     0;
    pRefCell                     0;
    pRefValue                    0;
    residualControl
    {
        p               1e-5;
        U               1e-5;
        nuTilda         1e-5;
    }
}
relaxationFactors
// Start with pRelax=0.3, URelax=0.7 and nuRelax=0.8
{
    fields
    {
        p               0.3;
    }
    equations
    {
        U               0.7;
        nuTilda         0.8;
    }
}
```

Appendix "C"

Summary of numerical results from the OpenFoam simulation

Description of four blades:

Blade #1: Circular top surface inscribed within 5.93" × 1.5" rectangle; flat bottom surface.
Blade #2: Circular top surface inscribed within 5.93" × 0.75" rectangle; flat botom surface.
Blade #3: Same as Blade#2, with convex bottom surface 0.25" thick.
Blade #4: Same as Blade#2, with concave bottom surface 0.25" thick.

Forces are stated in Newtons per meter of long axis. $\hat{X}$-direction is parallel to the relative wind far upstream; $\hat{Y}$-direction is perpendicular to the relative wind far upstream.

Blade	Speed	Angle of attack	Pressure forces		Viscous forces		Total forces		L/D ratio
			$\hat{X}$	$\hat{Y}$	$\hat{X}$	$\hat{Y}$	$\hat{X}$	$\hat{Y}$	
#1	4 fps	5°	9.4	91.1	2.1	0.3	11.5	91.4	7.9
#1	4 fps	15°	17.9	147.9	1.9	0	19.8	147.9	7.5
#1	6 fps	30°	128.8	327.4	0.9	-0.7	129.6	326.6	2.5
#2	4 fps	5°	3.6	103.9	2.1	0	5.7	103.9	18.2
#2	4 fps	15°	20.5	141.1	0.8	-0.2	21.3	140.8	6.6
#2	6 fps	30°	D.N.C.						
#2	4 fps	15°	18.1	113.6	1.0	-0.2	19.0	113.3	6.0
#2	4 fps	15°	23.1	164.5	0.7	-0.2	23.8	164.3	6.9

D.N.C. = did not converge.

The following table sets out the values of the dimensionless boundary layer thickness parameter y_+ for the various simulations.

Blade	Speed	Angle of attack	Top surface		Bottom surface	
			min y_+	max y_+	min y_+	max y_+
#1	4 fps	5°	1.0	3.9	0.3	6.4
#1	4 fps	15°	0.7	4.5	0.2	6.7
#1	6 fps	30°	0.2	7.5	0.2	7.5
#2	4 fps	5°	1.3	4.7	0.2	5.2
#2	4 fps	15°	0.5	5.4	0.1	4.9
#2	6 fps	30°	0.3	7.4	0.1	6.0
#2	4 fps	15°	0.4	6.9	0.1	4.9
#2	4 fps	15°	0.8	5.6	0.1	5.0

Appendix "D"

Listing of program `VB_YulohGMesh`

The following program was used to create a text file containing the instructions to be used by GMesh to create the geometry of the virtual wind tunnel. It consists of a main form named `Form1` and one module named `WriteGMeshFile`. It was developed in Visual Basic Express 2010. A screenshot of the GUI is shown after the listing.

```
Option Strict On
Option Explicit On

' Sets up the .geo file for a 2D steady-state viscous simulation of a yuloh profile.

Public Class Form1
    Inherits System.Windows.Forms.Form

    Public Sub New()
        InitializeComponent()
        With Me
            Name = ""
            Text = "2D yuloh profile in a steady waterflow"
            FormBorderStyle = Windows.Forms.FormBorderStyle.FixedSingle
            Size = New Drawing.Size(1024, 740)
            CenterToScreen()
            Visible = True
            Controls.Add(buttonExecute) : buttonExecute.BringToFront()
            Controls.Add(buttonExit) : buttonExit.BringToFront()
            Controls.Add(labelChord) : labelChord.BringToFront()
            Controls.Add(tbChord) : tbChord.BringToFront()
            Controls.Add(labelTopSurfThick) : labelTopSurfThick.BringToFront()
            Controls.Add(tbTopSurfThick) : tbTopSurfThick.BringToFront()
            Controls.Add(labelBotSurfThick) : labelBotSurfThick.BringToFront()
            Controls.Add(tbBotSurfThick) : tbBotSurfThick.BringToFront()
            Controls.Add(labelTotalThick) : labelTotalThick.BringToFront()
            Controls.Add(tbTotalThick) : tbTotalThick.BringToFront()
            Controls.Add(labelTopSurfRadius) : labelTopSurfRadius.BringToFront()
            Controls.Add(tbTopSurfRadius) : tbTopSurfRadius.BringToFront()
            Controls.Add(labelBotSurfRadius) : labelBotSurfRadius.BringToFront()
            Controls.Add(tbBotSurfRadius) : tbBotSurfRadius.BringToFront()
            Controls.Add(labelAngle) : labelAngle.BringToFront()
            Controls.Add(tbAngle) : tbAngle.BringToFront()
            Controls.Add(OutputArea) : OutputArea.BringToFront()
            PerformLayout()
        End With
        Initialization()
    End Sub

    '////////////////
    '// Data entry //
    '////////////////
    Public ChordEU As Double = 5.93             ' Length of chord, inches
    Public TopSurfThickEU As Double = 1.5       ' Top surface thickness, inches
    Public BotSurfThickEU As Double = 0         ' Bottom surface thickness, inches
    Public TotalThickEU As Double               ' Total airfoil thickness
    Public TopSurfRadiusEU As Double            ' Radius of top surface circle, inches
```

```
Public BotSurfRadiusEU As Double            ' Radius of bottom surface circle, inches
Public AngleAttackDeg As Double = 15        ' Angle of attack, degrees

'////////////////////////////////
'// Input variables in S.I. units //
'////////////////////////////////
Public ChordSI As Double                    ' Length of chord, meters
Public TopSurfThickSI As Double             ' Top surface thickness, meters
Public BotSurfThickSI As Double             ' Bottom surface thickness, meters
Public TotalThickSI As Double               ' Total airfoil thickness, meters
Public TopSurfRadiusSI As Double            ' Radius of top surface circle, meters
Public BotSurfRadiusSI As Double            ' Radius of bottom surface circle, meters
Public AngleAttackRad As Double             ' Angle of attack, radians
Public RelSpeedSI As Double                 ' Relative speed, meters per second

'//////////////////////////
'// Modeling paranmeters //
'//////////////////////////

Public NumSeg As Int32 = 1000               ' Number of segments on each surface
Public NumPoints As Int32 = NumSeg + 1      ' Number of points on each surafce

'////////////////////////////////
'// Definition of other variables //
'////////////////////////////////
Public Xtop(NumPoints) As Double            ' X-co-ordinates of points on top surface
Public Ytop(NumPoints) As Double            ' Y-co-ordinates of points on top surface
Public Xbot(NumPoints) As Double            ' X-co-ordinates of points on bottom
Public Ybot(NumPoints) As Double            ' Y-co-ordinates of points on bottom

'////////////////////////
'// Conversion factors //
'////////////////////////
Public MetersPerInch As Double = 2.54 / 100
Public MetersPerFoot As Double = 12 * 2.54 / 100
Public RadPerDeg As Double = Math.PI / 180

'////////////////////
'// Initialization //
'////////////////////
Public Sub Initialization()
    UpdateTheDisplayTextboxes()
    OutputArea.Text = ""
    Me.Refresh()
End Sub

'////////////////////////////////////////////////////////////////////////////////
'// Controls
'////////////////////////////////////////////////////////////////////////////////

Public labelChord As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(160, 20), _
     .Location = New Drawing.Point(5, 5), _
     .Text = "Chord (inch)", .TextAlign = ContentAlignment.MiddleLeft}

Public WithEvents tbChord As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(80, 20), _
     .Location = New Drawing.Point(170, 5), _
```

```
     .Text = "", .TextAlign = HorizontalAlignment.Left}

Public labelTopSurfThick As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(160, 20), _
     .Location = New Drawing.Point(5, 30), _
     .Text = "Top surface thickness (inch)", _
     .TextAlign = ContentAlignment.MiddleLeft}

Public WithEvents tbTopSurfThick As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(80, 20), _
     .Location = New Drawing.Point(170, 30), _
     .Text = "", .TextAlign = HorizontalAlignment.Left}

Public labelBotSurfThick As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(160, 20), _
     .Location = New Drawing.Point(5, 55), _
     .Text = "Bottom surface thickness (inch)", _
     .TextAlign = ContentAlignment.MiddleLeft}

Public WithEvents tbBotSurfThick As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(80, 20), _
     .Location = New Drawing.Point(170, 55), _
     .Text = "", .TextAlign = HorizontalAlignment.Left}

Public labelTotalThick As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(160, 20), _
     .Location = New Drawing.Point(5, 80), _
     .Text = "Total thickness (inch)", _
     .TextAlign = ContentAlignment.MiddleLeft}

Public tbTotalThick As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(80, 20), _
     .Location = New Drawing.Point(170, 80), _
     .Text = "", .TextAlign = HorizontalAlignment.Left, _
     .Enabled = False}

Public labelTopSurfRadius As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(160, 20), _
     .Location = New Drawing.Point(5, 105), _
     .Text = "Top surface radius (inch)", _
     .TextAlign = ContentAlignment.MiddleLeft}

Public tbTopSurfRadius As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(80, 20), _
     .Location = New Drawing.Point(170, 105), _
     .Text = "", .TextAlign = HorizontalAlignment.Left, _
     .Enabled = False}

Public labelBotSurfRadius As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(160, 20), _
     .Location = New Drawing.Point(5, 130), _
     .Text = "Bottom surface radius (inch)", _
     .TextAlign = ContentAlignment.MiddleLeft}

Public tbBotSurfRadius As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(80, 20), _
     .Location = New Drawing.Point(170, 130), _
     .Text = "", .TextAlign = HorizontalAlignment.Left, _
```

```
     .Enabled = False}

Public labelAngle As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(160, 20), _
     .Location = New Drawing.Point(5, 155), _
     .Text = "Angle (deg)", .TextAlign = ContentAlignment.MiddleLeft}

Public tbAngle As New Windows.Forms.TextBox With _
    {.Size = New Drawing.Size(80, 20), _
     .Location = New Drawing.Point(170, 155), _
     .Text = "", .TextAlign = HorizontalAlignment.Left}

Public WithEvents buttonExecute As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(245, 30), _
     .Location = New Drawing.Point(5, 180), _
     .Text = "Execute", _
     .TextAlign = ContentAlignment.MiddleCenter}

Public WithEvents buttonExit As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(245, 30), _
     .Location = New Drawing.Point(5, 215), _
     .Text = "Exit", _
     .TextAlign = ContentAlignment.MiddleCenter}

Public OutputArea As New Windows.Forms.Label With _
    {.Size = New Drawing.Size(600, 300), _
    .Location = New Drawing.Point(5, 250), _
    .TextAlign = ContentAlignment.TopLeft, _
    .BorderStyle = BorderStyle.FixedSingle}

'////////////////////////////////////////////////////////////////////////////////
'// Handlers for controls
'////////////////////////////////////////////////////////////////////////////////

Public Sub tbChord_Changed() Handles tbChord.TextChanged
    If (ChordEU <> Val(tbChord.Text)) Then
        ChordEU = Val(tbChord.Text)
        UpdateTheDisplayTextboxes()
        Me.Refresh()
    End If
End Sub

Public Sub tbTopSurfThick_Changed() Handles tbTopSurfThick.TextChanged
    If (TopSurfThickEU <> Val(tbTopSurfThick.Text)) Then
        TopSurfThickEU = Val(tbTopSurfThick.Text)
        UpdateTheDisplayTextboxes()
        Me.Refresh()
    End If
End Sub

Public Sub tbBotSurfThick_Changed() Handles tbBotSurfThick.TextChanged
    If (BotSurfThickEU <> Val(tbBotSurfThick.Text)) Then
        BotSurfThickEU = Val(tbBotSurfThick.Text)
        UpdateTheDisplayTextboxes()
        Me.Refresh()
    End If
End Sub
```

```
Public Sub buttonExecute_Click() Handles buttonExecute.MouseClick
    '
    ' Convert input variables into SI units
    ChordSI = ChordEU * MetersPerInch
    TopSurfThickSI = TopSurfThickEU * MetersPerInch
    BotSurfThickSI = BotSurfThickEU * MetersPerInch
    TotalThickSI = TotalThickEU * MetersPerInch
    TopSurfRadiusSI = TopSurfRadiusEU * MetersPerInch
    BotSurfRadiusSI = BotSurfRadiusEU * MetersPerInch
    AngleAttackRad = AngleAttackDeg * RadPerDeg
    '
    ' Calculate the points along the chord for offset co-ordinates
    ' The point (x, y) = (0, 0) is the center of the flat bottom face.
    Dim DeltaX As Double
    DeltaX = ChordSI / NumSeg
    For I As Int32 = 1 To NumPoints Step 1
        Xtop(I) = (-0.5 * ChordSI) + ((I - 1) * DeltaX)
        Xbot(I) = Xtop(I)
    Next I
    '
    ' Generate the vector of points along the top surface
    Dim Rtop As Double = TopSurfRadiusSI
    Dim Ttop As Double = TopSurfThickSI
    For I As Int32 = 1 To NumPoints Step 1
        Ytop(I) = Math.Sqrt((Rtop * Rtop) - (Xtop(I) * Xtop(I))) + Ttop - Rtop
    Next I
    '
    ' Generate the vector of points along the bottom surface
    Dim Rbot As Double = BotSurfRadiusSI
    Dim Tbot As Double = BotSurfThickSI
    For I As Int32 = 1 To NumPoints Step 1
        Ybot(I) = Math.Sqrt((Rbot * Rbot) - (Xbot(I) * Xbot(I))) + Tbot - Rbot
        ' Change algebraic sign to make concave (+) or convex (-) bottom.
        Ybot(I) = Ybot(I)
    Next I
    '
    ' Rotate the surfaces to the given angle of attack
    Dim SinAA As Double = Math.Sin(AngleAttackRad)
    Dim CosAA As Double = Math.Cos(AngleAttackRad)
    Dim Temp As Double
    For I As Int32 = 1 To NumPoints Step 1
        Temp = Xtop(I)
        Temp = (Xtop(I) * CosAA) + (Ytop(I) * SinAA)
        Ytop(I) = (Ytop(I) * CosAA) - (Xtop(I) * SinAA)
        Xtop(I) = Temp
        Temp = Xbot(I)
        Temp = (Xbot(I) * CosAA) + (Ybot(I) * SinAA)
        Ybot(I) = (Ybot(I) * CosAA) - (Xbot(I) * SinAA)
        Xbot(I) = Temp
    Next I
    '
    ' Write the GMesh file
    WriteGMeshFile.WriteGMeshFile(NumPoints, Xtop, Ytop, Xbot, Ybot)
    '
    ' Notify the user
    OutputArea.Text = "All done"
End Sub
```

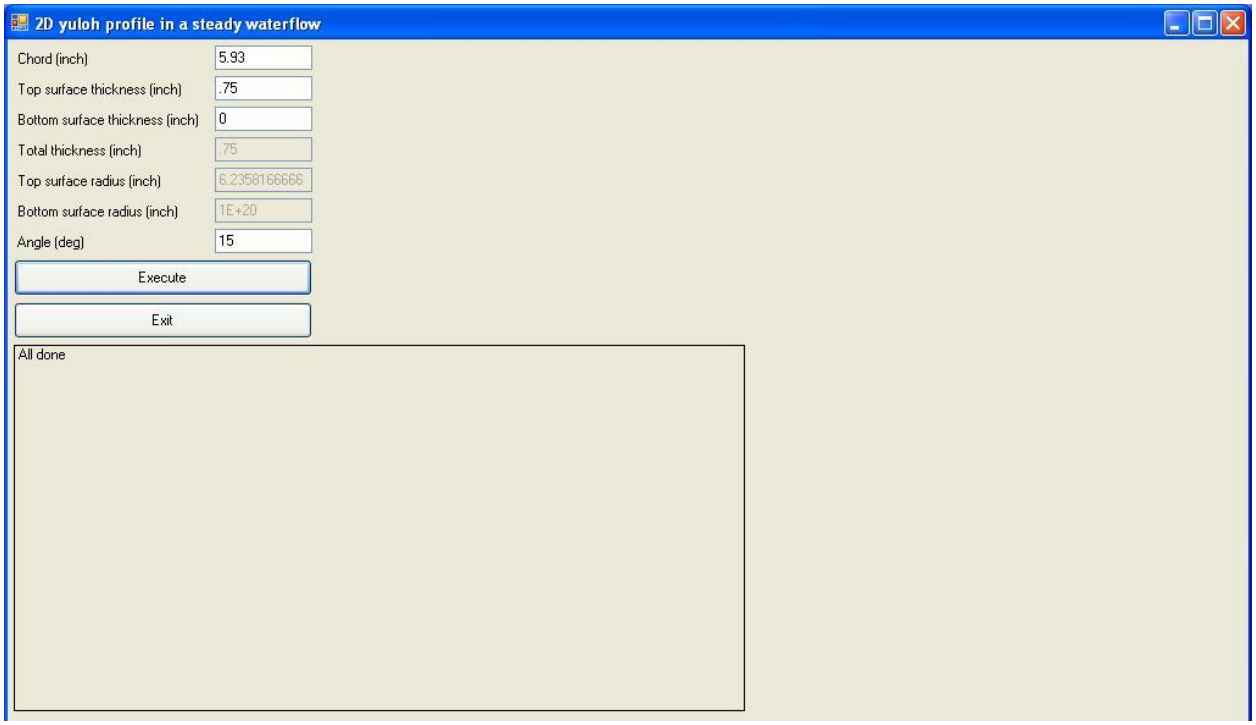
```
    Public Sub buttonExit_Click() Handles buttonExit.MouseClick
        Application.Exit()
    End Sub

    '////////////////////////////////////////////////////////////////////////////
    '// Calculation subroutines
    '////////////////////////////////////////////////////////////////////////////

    Public Function CalculateRadius( _
        ByVal lChord As Double, _
        ByVal lThick As Double) As Double
        ' This subroutine calculates the radius of a generating circle given a chord
        ' length and the thickness of the segment of the circle.
        If (lThick = 0) Then
            CalculateRadius = 1.0E+20
        Else
            CalculateRadius = (lChord * lChord / (8 * lThick)) + (lThick / 2)
        End If
    End Function

    Public Sub UpdateTheDisplayTextboxes()
        tbChord.Text = Trim(Str(ChordEU))
        tbTopSurfThick.Text = Trim(Str(TopSurfThickEU))
        tbBotSurfThick.Text = Trim(Str(BotSurfThickEU))
        TotalThickEU = TopSurfThickEU + BotSurfThickEU
        tbTotalThick.Text = Trim(Str(TotalThickEU))
        TopSurfRadiusEU = CalculateRadius(ChordEU, TopSurfThickEU)
        tbTopSurfRadius.Text = Trim(Str(TopSurfRadiusEU))
        BotSurfRadiusEU = CalculateRadius(ChordEU, BotSurfThickEU)
        tbBotSurfRadius.Text = Trim(Str(BotSurfRadiusEU))
        tbAngle.Text = Trim(Str(AngleAttackDeg))
    End Sub

End Class




Option Strict On
Option Explicit On

Public Module WriteGMeshFile

    ' The X-Y co-ordinates of the surfaces which are passed as arguments to the
    ' subroutine in this module include the rotation due to the angle of attack.

    '////////////////////////////
    '// Wind tunnel parameters //
    '////////////////////////////
    ' The wind tunnel ("WT") is positioned with respect to the base center of the airfoil
    Public WTDistanceAhead As Double = 2        ' Distance from O to Inlet, meters
    Public WTDistanceAstern As Double = 2       ' Distance from O to Outlet, meters
    Public WTDistanceAbove As Double = 2        ' Distance from O to top of WT, meters
    Public WTDistanceBelow As Double = 2        ' Distance from O to bottom of WT, meters
    Public lcWT As Double = 0.025               ' Size of mesh on WT, meters
    Public lcYuloh As Double = 0.00025          ' Size of mesh on Yuloh, meters
    Public WTHalfThick As Double = 0.0005       ' Half-thickness of WT, mm
```

```
Public FileName As String = "Yuloh.geo.txt" ' Name of output file
Public Filewriter As System.IO.StreamWriter

'////////////////////////////////
'// Definition of other variables //
'////////////////////////////////
' Yuloh reference indices
Public FirstPt As Int32                     ' Index of first Point
Public LastPt As Int32                      ' Index of last Point
Public FirstLn As Int32                     ' Index of first Line
Public LastLn As Int32                      ' Index of last Line
Public AirfoilLineLoop As Int32             ' Index of LineLoop around airfoil
' Wind tunnel reference indices
Public FirstPtOnWT As Int32                 ' Index of first Point on wind tunnel
Public FirstLnAlngWT As Int32               ' Index of first Line around WT
Public WTLineLoop As Int32                  ' Index of Line Loop around wind tunnel
' Surface Loop reference indices
Public WTSurface As Int32                   ' Index of WT Plane Surface, right side
' Other variables
Public AngleAttackRad As Double


Public Sub WriteGMeshFile( _
    ByVal NumPoints As Int32, _
    ByVal Xtop() As Double, ByVal Ytop() As Double, _
    ByVal Xbot() As Double, ByVal Ybot() As Double)
    '
    ' Step #1: Open the output file
    Filewriter = New System.IO.StreamWriter(FileName)
    '
    ' Step #2: Write header information to the output file
    Filewriter.Write( _
        "// Yuloh in 2D waterflow" & vbCrLf & _
        "Mesh.RandomFactor = 1e-11;" & vbCrLf & _
        "Geometry.AutoCoherence = 1;" & vbCrLf & _
        "Geometry.HighlightOrphans = 1;" & vbCrLf & _
        "Geometry.MatchGeomAndMesh = 1;" & vbCrLf & _
        "Geometry.SnapX = 0;" & vbCrLf & _
        "Geometry.SnapY = 0;" & vbCrLf & _
        "Geometry.SnapZ = 0;" & vbCrLf & _
        "Geometry.Tolerance = 1e-15;" & vbCrLf & vbCrLf & _
        "WTDistanceAhead = " & Trim(Str(WTDistanceAhead)) & ";" & vbCrLf & _
        "WTDistanceAstern = " & Trim(Str(WTDistanceAstern)) & ";" & vbCrLf & _
        "WTDistanceAbove = " & Trim(Str(WTDistanceAbove)) & ";" & vbCrLf & _
        "WTDistanceBelow = " & Trim(Str(WTDistanceBelow)) & ";" & vbCrLf & _
        "lcWT = " & Trim(Str(lcWT)) & ";" & vbCrLf & _
        "lcYuloh = " & Trim(Str(lcYuloh)) & ";" & vbCrLf & _
        "WTHalfThick = " & Trim(Str(WTHalfThick)) & ";" & vbCrLf)
    '
    ' Step #3: Points around the airfoil, right side, clockwise from the LE
    FirstPt = 1
    LastPt = 0
    Filewriter.Write( _
        "//" & vbCrLf & _
        "// Points around the airfoil, right side, clockwise from the LE" & vbCrLf)
    For I As Int32 = 1 To NumPoints Step 1
        LastPt = LastPt + 1
        Filewriter.Write( _
```

```
        "Point(" & Trim(Str(LastPt)) & ") = {" & _
        FormatNumber(Xtop(I), 12) & ", " & _
        FormatNumber(Ytop(I), 12) & ", " & _
        "-WTHalfThick, lcYuloh};" & vbCrLf)
Next I
For I As Int32 = (NumPoints - 1) To 2 Step -1
    LastPt = LastPt + 1
    Filewriter.Write( _
        "Point(" & Trim(Str(LastPt)) & ") = {" & _
        FormatNumber(Xbot(I), 12) & ", " & _
        FormatNumber(Ybot(I), 12) & ", " & _
        "-WTHalfThick, lcYuloh};" & vbCrLf)
Next I
'
' Step #4: Lines around the airfoil, right side, clockwise from the LE
FirstLn = LastPt + 1
LastLn = LastPt
Filewriter.Write( _
    "//" & vbCrLf & _
    "// Lines around the airfoil, right side, clockwise from LE" & vbCrLf)
For I As Int32 = 1 To (2 * (NumPoints - 1)) Step 1
    Dim PointIndexFrom As Int32
    Dim PointIndexTo As Int32
    LastLn = LastLn + 1
    PointIndexFrom = FirstPt + I - 1
    If (I <> (2 * (NumPoints - 1))) Then
        PointIndexTo = FirstPt + I
    Else
        PointIndexTo = FirstPt
    End If
    Filewriter.Write( _
        "Line(" & Trim(Str(LastLn)) & _
        ") = {" & Trim(Str(PointIndexFrom)) & _
        ", " & Trim(Str(PointIndexTo)) & "};" & vbCrLf)
Next I
'
' Step #5: Line Loop around the airfoil, right side, directed outwards
Filewriter.Write( _
    "//" & vbCrLf & _
    "// Line Loop around the airfoil, right side, directed outwards" & vbCrLf)
AirfoilLineLoop = LastLn + 1
Dim NumbersAcrossPage As Int32 = 11
Filewriter.Write("Line Loop(" & Trim(Str(AirfoilLineLoop)) & ") = {")
For I As Int32 = 1 To (2 * (NumPoints - 1)) Step 1
    Dim LineIndex As Int32 = FirstLn + I - 1
    If (I <> (2 * (NumPoints - 1))) Then
        If (NumbersAcrossPage > 10) Then
            Filewriter.Write(vbCrLf & "     " & Trim(Str(LineIndex)) & ",")
            NumbersAcrossPage = 1
        Else
            Filewriter.Write(Trim(Str(LineIndex)) & ",")
            NumbersAcrossPage = NumbersAcrossPage + 1
        End If
    Else
        Filewriter.Write(Trim(Str(LineIndex)) & "};" & vbCrLf)
    End If
Next I
'
```

```
' Step #6: Points at the corners of the wind tunnel
Filewriter.Write( _
     "//" & vbCrLf & _
     "// Points at the corners of the wind tunnel" & vbCrLf)
FirstPtOnWT = LastPt + 1
Filewriter.Write("Point(" & Trim(Str(FirstPtOnWT)) & ") = " & _
    "{-WTDistanceAhead, WTDistanceAbove, -WTHalfThick, lcWT};" & vbCrLf)
Filewriter.Write("Point(" & Trim(Str(FirstPtOnWT + 1)) & ") = " & _
    "{WTDistanceAstern, WTDistanceAbove, -WTHalfThick, lcWT};" & vbCrLf)
Filewriter.Write("Point(" & Trim(Str(FirstPtOnWT + 2)) & ") = " & _
    "{WTDistanceAstern, -WTDistanceBelow, -WTHalfThick, lcWT};" & vbCrLf)
Filewriter.Write("Point(" & Trim(Str(FirstPtOnWT + 3)) & ") = " & _
    "{-WTDistanceAhead, -WTDistanceBelow, -WTHalfThick, lcWT};" & vbCrLf)
'
' Step #7: Lines along the edges of the wind tunnel, clockwise
Filewriter.Write( _
     "//" & vbCrLf & _
     "// Lines along the edges of the wind tunnel, clockwise" & vbCrLf)
FirstLnAlngWT = AirfoilLineLoop + 1
Filewriter.Write("Line(" & Trim(Str(FirstLnAlngWT)) & ") = {" & _
    Trim(Str(FirstPtOnWT)) & ", " & _
    Trim(Str(FirstPtOnWT + 1)) & "};" & vbCrLf)
Filewriter.Write("Line(" & Trim(Str(FirstLnAlngWT + 1)) & ") = {" & _
    Trim(Str(FirstPtOnWT + 1)) & ", " & _
    Trim(Str(FirstPtOnWT + 2)) & "};" & vbCrLf)
Filewriter.Write("Line(" & Trim(Str(FirstLnAlngWT + 2)) & ") = {" & _
    Trim(Str(FirstPtOnWT + 2)) & ", " & _
    Trim(Str(FirstPtOnWT + 3)) & "};" & vbCrLf)
Filewriter.Write("Line(" & Trim(Str(FirstLnAlngWT + 3)) & ") = {" & _
    Trim(Str(FirstPtOnWT + 3)) & ", " & _
    Trim(Str(FirstPtOnWT)) & "};" & vbCrLf)

' Step #8: Line Loop around the wind tunnel, directed outwards
Filewriter.Write( _
     "//" & vbCrLf & _
     "// Line Loop around the wind tunnel, directed outwards" & vbCrLf)
WTLineLoop = FirstLnAlngWT + 4
Filewriter.Write("Line Loop(" & Trim(Str(WTLineLoop)) & ") = {" & _
     Trim(Str(FirstLnAlngWT)) & ", " & _
     Trim(Str(FirstLnAlngWT + 1)) & ", " & _
     Trim(Str(FirstLnAlngWT + 2)) & ", " & _
     Trim(Str(FirstLnAlngWT + 3)) & "};" & vbCrLf)

' Step #9: Plane Surface on the wind tunnel, right side, directed outwards
Filewriter.Write( _
     "//" & vbCrLf & _
     "// Plane Surface on the wind tunnel, right side," & vbCrLf & _
     "// excluding the hole left by the membrane." & vbCrLf)
WTSurface = WTLineLoop + 1
Filewriter.Write("Plane Surface(" & Trim(Str(WTSurface)) & ") = {" & _
    Trim(Str(WTLineLoop)) & ", " & Trim(Str(AirfoilLineLoop)) & "};" & vbCrLf)

'////////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////
'// Extrusion of the right-hand side into the left-hand side ////////////////
''//////////////////////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////
```

```
        ' Step #10: Extrude the Plane Surface of the wind tunnel in the Z-direction
        Filewriter.Write( _
            "//" & vbCrLf & _
            "// Extrude Plane Surface of the wind tunnel in the Z-direction" & vbCrLf)
        Filewriter.Write("NewWT[] = " & _
            "Extrude { 0 , 0 , 2 * WTHalfThick } {" & vbCrLf & _
            "    Surface{" & Trim(Str(WTSurface)) & "};" & vbCrLf & _
            "    Layers{1};" & vbCrLf & _
            "    Recombine; };" & vbCrLf)

        '////////////////////////////////////////////////////////////////////////////
        '////////////////////////////////////////////////////////////////////////////
        '// Define Physical Surfaces ////////////////////////////////////////////////
        ''//////////////////////////////////////////////////////////////////////////
        '////////////////////////////////////////////////////////////////////////////
        '
        ' Step #11: Define Physical Surfaces for OpenFoam's use
        Filewriter.Write( _
            "//" & vbCrLf & _
            "// Physical Surfaces on Yuloh for OpenFoam's use" & vbCrLf)
        For I As Int32 = 1 To (2 * (NumPoints - 1)) Step 1
            Filewriter.Write( _
                "Physical Surface(""Segment." & Trim(Str(I)) & _
                """) = { NewWT[" & Trim(Str(5 + I)) & "] };" & vbCrLf)
        Next I
        '
        ' Step #12: Define Physical Surfaces on the wind tunnel for OpenFoam's use
        Filewriter.Write( _
             "//" & vbCrLf & _
             "// Physical Surfaces on the wind tunnel for OpenFoam's use" & vbCrLf)
        Filewriter.Write( _
            "Physical Surface(""LeftWall"") = { NewWT[0] };" & vbCrLf & _
            "Physical Surface(""Top"") = { NewWT[2] };" & vbCrLf & _
            "Physical Surface(""Outlet"") = { NewWT[3] };" & vbCrLf & _
            "Physical Surface(""Bottom"") = { NewWT[4] };" & vbCrLf & _
            "Physical Surface(""Inlet"") = { NewWT[5] };" & vbCrLf & _
            "Physical Surface(""RightWall"") = { " & _
            Trim(Str(WTSurface)) & " };" & vbCrLf)
        '
        ' Step #13: Define the Physical Volume for OpenFoam's use
        Filewriter.Write( _
             "//" & vbCrLf & _
             "// Define the Physical Volume for OpenFoam's use" & vbCrLf)
        Filewriter.Write("Physical Volume(""Internal"") = { NewWT[1] };" & vbCrLf)
        '
        ' Step #14: Conclude
        Filewriter.Close()
    End Sub

End Module
```


2D yuloh profile in a steady waterflow
Chord (inch)
5.93
Top surface thickness (inch)
.75
Bottom surface thickness (inch)
0
Total thickness (inch)
.75
Top surface radius (inch)
6.2358166666
Bottom surface radius (inch)
1E+20
Angle (deg)
15
Execute
Exit
All done

Appendix "E"

Listing of program `VB_YulohKinematics`

The following program was used to calculate and plot the kinematics of the blade's trajectory. It consists of a main form named `Form1` and two modules. The module named `Trajectories` calculates the trajectories of a line segment across the flat side of the blade at a given cross-section. This is the routine which produces the "falling leaf"-type plots. The module named `AnglesOfAttack` calculates the relative wind at points along the long axis of the blade. Both modules plot their results on a bitmap which occupies most the the screen. The main form has two buttons. They execute the calculations. For each button, there are subordinate buttons which determine the geometric plane used to display the results. The program was developed in Visual Basic Express 2010. A screenshot of the GUI is shown after the listing.

```
Option Strict On
Option Explicit On

' Calculates the time-trajectories of points on a Yuloh's blade

Public Class Form1
    Inherits System.Windows.Forms.Form

    Public Sub New()
        InitializeComponent()
        With Me
            Name = ""
            Text = "Trajectories of points on a Yuloh"
            FormBorderStyle = Windows.Forms.FormBorderStyle.FixedSingle
            Size = New Drawing.Size(1024, 740)
            CenterToScreen()
            Visible = True
            Controls.Add(buttonCalculateTrajectories)
            buttonCalculateTrajectories.BringToFront()
            Controls.Add(buttonPlotXY) : buttonPlotXY.BringToFront()
            Controls.Add(buttonPlotYZ) : buttonPlotYZ.BringToFront()
            Controls.Add(buttonPlotXZ) : buttonPlotXZ.BringToFront()
            Controls.Add(buttonCalculateAngles)
            buttonCalculateAngles.BringToFront()
            Controls.Add(buttonPlot1Frame) : buttonPlot1Frame.BringToFront()
            Controls.Add(buttonPlot6Frame) : buttonPlot6Frame.BringToFront()
            Controls.Add(buttonPlot6XZFrame) : buttonPlot6XZFrame.BringToFront()
            Controls.Add(buttonPlot6YZFrame) : buttonPlot6YZFrame.BringToFront()
            Controls.Add(PlotArea) : PlotArea.BringToFront()
            Controls.Add(TextArea) : TextArea.BringToFront()
            PerformLayout()
        End With
        Initialization()
    End Sub

    Public TextArea As New Windows.Forms.Label With _
        {.Size = New Drawing.Size(950, 400), _
         .Location = New Drawing.Point(10, 50)}

    '////////////////////
    '// Initialization //
    '////////////////////
```

```
Public Sub Initialization()
End Sub

'////////////////////////////////////////////////////////////////////////////////
'// Controls
'////////////////////////////////////////////////////////////////////////////////

Public WithEvents buttonCalculateTrajectories As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(150, 30), _
     .Location = New Drawing.Point(5, 5), _
     .Text = "Calculate trajectories", _
     .TextAlign = ContentAlignment.MiddleCenter}

Public WithEvents buttonPlotXY As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(80, 30), _
     .Location = New Drawing.Point(160, 5), _
     .Text = "Plot XY plane", _
     .TextAlign = ContentAlignment.MiddleCenter, _
     .Enabled = False}

Public WithEvents buttonPlotYZ As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(80, 30), _
     .Location = New Drawing.Point(245, 5), _
     .Text = "Plot YZ plane", _
     .TextAlign = ContentAlignment.MiddleCenter, _
     .Enabled = False}

Public WithEvents buttonPlotXZ As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(80, 30), _
     .Location = New Drawing.Point(330, 5), _
     .Text = "Plot XZ plane", _
     .TextAlign = ContentAlignment.MiddleCenter, _
     .Enabled = False}

Public WithEvents buttonCalculateAngles As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(150, 30), _
     .Location = New Drawing.Point(415, 5), _
     .Text = "Calculate angles of attack", _
     .TextAlign = ContentAlignment.MiddleCenter}

Public WithEvents buttonPlot1Frame As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(80, 30), _
     .Location = New Drawing.Point(570, 5), _
     .Text = "Plot 1 frame", _
     .TextAlign = ContentAlignment.MiddleCenter, _
     .Enabled = False}

Public WithEvents buttonPlot6Frame As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(80, 30), _
     .Location = New Drawing.Point(655, 5), _
     .Text = "Plot 6 frame", _
     .TextAlign = ContentAlignment.MiddleCenter, _
     .Enabled = False}

Public WithEvents buttonPlot6XZFrame As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(80, 30), _
     .Location = New Drawing.Point(740, 5), _
     .Text = "Plot 6XZ", _
```

```
     .TextAlign = ContentAlignment.MiddleCenter, _
     .Enabled = False}

Public WithEvents buttonPlot6YZFrame As New Windows.Forms.Button With _
    {.Size = New Drawing.Size(80, 30), _
     .Location = New Drawing.Point(825, 5), _
     .Text = "Plot 6YZ", _
     .TextAlign = ContentAlignment.MiddleCenter, _
     .Enabled = False}

Public PlotArea As New Windows.Forms.Panel With _
    {.Size = New Drawing.Size(1000, 650), _
    .Location = New Drawing.Point(5, 40), _
    .BorderStyle = BorderStyle.FixedSingle}

Public PlotBitmap As New Bitmap(1000, 650)

'////////////////////////////////////////////////////////////////////////////////
'// Handlers
'////////////////////////////////////////////////////////////////////////////////

Public Sub buttonCalculateTrajectories_Click() Handles _
    buttonCalculateTrajectories.MouseClick
    CalculateTrajectories()
    buttonPlotXZ.Enabled = True
    buttonPlotYZ.Enabled = True
    buttonPlotXY.Enabled = True
End Sub

Public Sub buttonPlotXY_Click() Handles buttonPlotXY.MouseClick
    ' Part A: Clear the graphics
    Dim g As Graphics = Graphics.FromImage(PlotBitmap)
    g.Clear(Color.White)
    g.Dispose()
    PlotArea.BackgroundImage = PlotBitmap
    PlotArea.Refresh()
    ' Part B: Paint the Bitmap
    Dim e As System.EventArgs
    RenderTrajectories( _
        PlotArea, e, PlotBitmap, "XY")
    ' Part C: Display the Bitmap
    PlotArea.BackgroundImage = PlotBitmap
    PlotArea.Refresh()
End Sub

Public Sub buttonPlotYZ_Click() Handles buttonPlotYZ.MouseClick
    ' Part A: Clear the graphics
    Dim g As Graphics = Graphics.FromImage(PlotBitmap)
    g.Clear(Color.White)
    g.Dispose()
    PlotArea.BackgroundImage = PlotBitmap
    PlotArea.Refresh()
    ' Part B: Paint the Bitmap
    Dim e As System.EventArgs
    RenderTrajectories( _
        PlotArea, e, PlotBitmap, "YZ")
    ' Part C: Display the Bitmap
    PlotArea.BackgroundImage = PlotBitmap
```

```
    PlotArea.Refresh()
End Sub

Public Sub buttonPlotXZ_Click() Handles buttonPlotXZ.MouseClick
    ' Part A: Clear the graphics
    Dim g As Graphics = Graphics.FromImage(PlotBitmap)
    g.Clear(Color.White)
    g.Dispose()
    PlotArea.BackgroundImage = PlotBitmap
    PlotArea.Refresh()
    ' Part B: Paint the Bitmap
    Dim e As System.EventArgs
    RenderTrajectories( _
        PlotArea, e, PlotBitmap, "XZ")
    ' Part C: Display the Bitmap
    PlotArea.BackgroundImage = PlotBitmap
    PlotArea.Refresh()
End Sub

Public Sub buttonCalculateAngles_Click() Handles buttonCalculateAngles.MouseClick
    CalculateAnglesOfAttack()
    buttonPlot1Frame.Enabled = True
    buttonPlot6Frame.Enabled = True
    buttonPlot6XZFrame.Enabled = True
    buttonPlot6YZFrame.Enabled = True
End Sub

Public Sub buttonPlot1Frame_Click() Handles buttonPlot1Frame.MouseClick
    ' Part A: Clear the graphics
    Dim g As Graphics = Graphics.FromImage(PlotBitmap)
    g.Clear(Color.White)
    g.Dispose()
    PlotArea.BackgroundImage = PlotBitmap
    PlotArea.Refresh()
    ' Part B: Paint the Bitmap
    Dim e As System.EventArgs
    RenderAnglesOfAttackIn1Frame( _
        PlotArea, e, PlotBitmap)
    ' Part C: Display the Bitmap
    PlotArea.BackgroundImage = PlotBitmap
    PlotArea.Refresh()
End Sub

Public Sub buttonPlot6Frame_Click() Handles buttonPlot6Frame.MouseClick
    ' Part A: Clear the graphics
    Dim g As Graphics = Graphics.FromImage(PlotBitmap)
    g.Clear(Color.White)
    g.Dispose()
    PlotArea.BackgroundImage = PlotBitmap
    PlotArea.Refresh()
    ' Part B: Paint the Bitmap
    Dim e As System.EventArgs
    RenderAnglesOfAttackIn6Frame( _
        PlotArea, e, PlotBitmap)
    ' Part C: Display the Bitmap
    PlotArea.BackgroundImage = PlotBitmap
    PlotArea.Refresh()
End Sub
```

```
    Public Sub buttonPlot6XZFrame_Click() Handles buttonPlot6XZFrame.MouseClick
        ' Part A: Clear the graphics
        Dim g As Graphics = Graphics.FromImage(PlotBitmap)
        g.Clear(Color.White)
        g.Dispose()
        PlotArea.BackgroundImage = PlotBitmap
        PlotArea.Refresh()
        ' Part B: Paint the Bitmap
        Dim e As System.EventArgs
        RenderAnglesOfAttackIn6XZFrame( _
            PlotArea, e, PlotBitmap)
        ' Part C: Display the Bitmap
        PlotArea.BackgroundImage = PlotBitmap
        PlotArea.Refresh()
    End Sub

    Public Sub buttonPlot6YZFrame_Click() Handles buttonPlot6YZFrame.MouseClick
        ' Part A: Clear the graphics
        Dim g As Graphics = Graphics.FromImage(PlotBitmap)
        g.Clear(Color.White)
        g.Dispose()
        PlotArea.BackgroundImage = PlotBitmap
        PlotArea.Refresh()
        ' Part B: Paint the Bitmap
        Dim e As System.EventArgs
        RenderAnglesOfAttackIn6yZFrame( _
            PlotArea, e, PlotBitmap)
        ' Part C: Display the Bitmap
        PlotArea.BackgroundImage = PlotBitmap
        PlotArea.Refresh()
    End Sub

End Class


Option Strict On
Option Explicit On

Public Module Trajectories

    '////////////////
    '// Data entry //
    '////////////////
    Private D As Double = 9                     ' Translation down the Yuloh shaft, feet
    Private PsiDeg As Double = 40               ' Slant angle, deg
    Private ThetaMax As Double = 12.4           ' Maximum side-to-side deflection, deg
    Private PhiMax As Double = 45               ' Maximum loom twist angle, deg
    Private V As Double = 2.35                  ' Boat speed, feet per second
    Private THStroke As Double = 1              ' Duration of half-stroke, seconds
    Private Ax6 As Double = 0                   ' X6-co-ordinates of point A
    Private Ay6 As Double = -0.0417             ' Y6-co-ordinates of point A
    Private Az6 As Double = +0.146              ' Z6-co-ordinates of point A
    Private Bx6 As Double = 0                   ' X6-co-ordinates of point B
    Private By6 As Double = -0.0417             ' Y6-co-ordinates of point B
    Private Bz6 As Double = -0.146              ' Z6-co-ordinates of point B
    Private NPerHStroke As Int32 = 1000         ' Number of time steps per half stroke
    Private NHStrokes As Int32 = 5              ' Number of half-strokes to simulate
```

```
'//////////////////////
'// Stroke variables //
'//////////////////////
Private CurrentStrk As String               ' "P" or "S"
Private ThetaDot As Double = 2 * ThetaMax / THStroke ' Stroke speed, deg per sec
Private TimeInStroke As Double              ' Time since start of stroke, seconds

'///////////////////////////////////
'// Definition of other variables //
'///////////////////////////////////
Private delT As Double = THStroke / NPerHStroke   ' Duration of time step, seconds
Private NTotal As Int32 = NPerHStroke * NHStrokes ' Number of time steps
Private NCurrent As Int32                   ' Number of current time step
Private PsiRad As Double                    ' Slant angle, radians
Private ThetaDeg As Double                  ' Side-to-side deflection, degrees
Private ThetaRad As Double                  ' Side-to-side deflection, radians
Private PhiDeg As Double                    ' Loom twist angle, degrees
Private PhiRad As Double                    ' Loom twist angle, radians
Private cosPsi, sinPsi As Double            ' Trigonometric function
Private cosTheta, sinTheta As Double        ' "
Private cosPhi, sinPhi As Double            ' "
Private T As Double                         ' Time, seconds
Private Ax1(NTotal) As Double               ' X1-co-ordinates of point A
Private Ay1(NTotal) As Double               ' Y1-co-ordinates of point A
Private Az1(NTotal) As Double               ' Z1-co-ordinates of point A
Private Bx1(NTotal) As Double               ' X1-co-ordinates of point B
Private By1(NTotal) As Double               ' Y1-co-ordinates of point B
Private Bz1(NTotal) As Double               ' Z1-co-ordinates of point B

Public Sub CalculateTrajectories()
    ' Intermediate matrix products
    Dim Ax5, Bx5 As Double
    Dim Ay5, By5 As Double
    Dim Az5, Bz5 As Double
    Dim Ax4, Bx4 As Double
    Dim Ay4, By4 As Double
    Dim Az4, Bz4 As Double
    Dim Ax3, Bx3 As Double
    Dim Ay3, By3 As Double
    Dim Az3, Bz3 As Double
    Dim Ax2, Bx2 As Double
    Dim Ay2, By2 As Double
    Dim Az2, Bz2 As Double
    ' Set initial conditions
    CurrentStrk = "P"
    NCurrent = 0
    For Istroke As Int32 = 1 To NHStrokes Step 1
        ' Change sense of half stroke
        If (CurrentStrk = "P") Then
            CurrentStrk = "S"
        Else
            CurrentStrk = "P"
        End If
        For Jstep As Int32 = 0 To (NPerHStroke - 1)
            ' Determine time in this half-stroke
            TimeInStroke = Jstep * delT
            ' Increment the master clock and the number of the current time step
```

```
            T = T + delT
            NCurrent = NCurrent + 1
            ' Calculate angles Theta and Phi, in degrees
            If (CurrentStrk = "S") Then
                ThetaDeg = -ThetaMax + (ThetaDot * TimeInStroke)
                PhiDeg = +PhiMax
            Else
                ThetaDeg = +ThetaMax - (ThetaDot * TimeInStroke)
                PhiDeg = -PhiMax
            End If
            ' Convert all angles to radians
            PsiRad = PsiDeg * Math.PI / 180
            ThetaRad = ThetaDeg * Math.PI / 180
            PhiRad = PhiDeg * Math.PI / 180
            ' Compute the trigonometric functions
            cosPsi = Math.Cos(PsiRad)
            sinPsi = Math.Sin(PsiRad)
            cosTheta = Math.Cos(ThetaRad)
            sinTheta = Math.Sin(ThetaRad)
            cosPhi = Math.Cos(PhiRad)
            sinPhi = Math.Sin(PhiRad)
            ' Transform from frame 6 to frame 5
            Ax5 = Ax6 - D
            Ay5 = Ay6
            Az5 = Az6
            Bx5 = Bx6 - D
            By5 = By6
            Bz5 = Bz6
            ' Transform from frame 5 to frame 4
            Ax4 = Ax5
            Ay4 = (cosPhi * Ay5) + (-sinPhi * Az5)
            Az4 = (sinPhi * Ay5) + (cosPhi * Az5)
            Bx4 = Bx5
            By4 = (cosPhi * By5) + (-sinPhi * Bz5)
            Bz4 = (sinPhi * By5) + (cosPhi * Bz5)
            ' Transform from frame 4 to frame 3
            Ax3 = (cosTheta * Ax4) + (sinTheta * Az4)
            Ay3 = Ay4
            Az3 = (-sinTheta * Ax4) + (cosTheta * Az4)
            Bx3 = (cosTheta * Bx4) + (sinTheta * Bz4)
            By3 = By4
            Bz3 = (-sinTheta * Bx4) + (cosTheta * Bz4)
            ' Transform from frame 3 to frame 2
            Ax2 = (cosPsi * Ax3) + (-sinPsi * Ay3)
            Ay2 = (sinPsi * Ax3) + (cosPsi * Ay3)
            Az2 = Az3
            Bx2 = (cosPsi * Bx3) + (-sinPsi * By3)
            By2 = (sinPsi * Bx3) + (cosPsi * By3)
            Bz2 = Bz3
            ' Transform from frame 2 to frame 1
            Ax1(NCurrent) = Ax2 + (V * T)
            Ay1(NCurrent) = Ay2
            Az1(NCurrent) = Az2
            Bx1(NCurrent) = Bx2 + (V * T)
            By1(NCurrent) = By2
            Bz1(NCurrent) = Bz2
        Next Jstep
    Next Istroke
```

```
End Sub

Public Sub RenderTrajectories( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs, _
    ByRef PlotBitmap As Bitmap, _
    ByVal Plane As String)
    ' Find the extreme X- and Y- and Z-values, in feet
    Dim xMax As Double = -1.0E+20
    Dim xMin As Double = 1.0E+20
    Dim yMax As Double = -1.0E+20
    Dim yMin As Double = 1.0E+20
    Dim zMax As Double = -1.0E+20
    Dim zMin As Double = 1.0E+20
    For I As Int32 = 1 To NTotal Step 1
        If (Ax1(I) > xMax) Then xMax = Ax1(I)
        If (Ax1(I) < xMin) Then xMin = Ax1(I)
        If (Ay1(I) > yMax) Then yMax = Ay1(I)
        If (Ay1(I) < yMin) Then yMin = Ay1(I)
        If (Az1(I) > zMax) Then zMax = Az1(I)
        If (Az1(I) < zMin) Then zMin = Az1(I)
        If (Bx1(I) > xMax) Then xMax = Bx1(I)
        If (Bx1(I) < xMin) Then xMin = Bx1(I)
        If (By1(I) > yMax) Then yMax = By1(I)
        If (By1(I) < yMin) Then yMin = By1(I)
        If (Bz1(I) > zMax) Then zMax = Bz1(I)
        If (Bz1(I) < zMin) Then zMin = Bz1(I)
    Next I
    ' Find the extreme overall distance
    Dim MaxDistance As Double = -1.0E+20
    If ((xMax - xMin) > MaxDistance) Then
        MaxDistance = xMax - xMin
    End If
    If ((yMax - yMin) > MaxDistance) Then
        MaxDistance = yMax - yMin
    End If
    If ((zMax - zMin) > MaxDistance) Then
        MaxDistance = zMax - zMin
    End If
    ' Calculate the appropriate scaling factor, in pixels per foot
    ' Leave a 5% margin all around the display.
    Dim SFPixelsPerFoot As Double
    SFPixelsPerFoot = 1000 / (1.1 * MaxDistance)
    ' Express the location and offset of the bitmap in feet
    Dim bmLeftFeet As Double
    Dim bmTopFeet As Double
    Select Case Plane
        Case "XY"
            bmLeftFeet = xMin - (0.05 * MaxDistance)
            bmTopFeet = Math.Max(0, yMax) + (0.05 * MaxDistance)
        Case "YZ"
            bmLeftFeet = zMin - (0.05 * MaxDistance)
            bmTopFeet = Math.Max(0, yMax) + (0.05 * MaxDistance)
        Case "XZ"
            bmLeftFeet = xMin - (0.05 * MaxDistance)
            bmTopFeet = zMax + (0.05 * MaxDistance)
    End Select
    ' Define the graphics object
```

```
Dim g As Graphics = Graphics.FromImage(PlotBitmap)
Dim PlotPen As New Drawing.Pen(Color.Red, 2)
Dim AxisPen As New Drawing.Pen(Color.Black, 2)
' Draw the segments one-by-one starting from the first time step
Dim StartX As Double
Dim StartY As Double
Dim StopX As Double
Dim StopY As Double
For I As Int32 = 1 To NTotal Step 100
    Select Case Plane
        Case "XY"
            StartX = (Ax1(I) - bmLeftFeet) * SFPixelsPerFoot
            StartY = (bmTopFeet - Ay1(I)) * SFPixelsPerFoot
            StopX = (Bx1(I) - bmLeftFeet) * SFPixelsPerFoot
            StopY = (bmTopFeet - By1(I)) * SFPixelsPerFoot
            g.DrawLine(PlotPen, _
                CSng(StartX), CSng(StartY), CSng(StopX), CSng(StopY))
        Case "YZ"
            StartX = (Az1(I) - bmLeftFeet) * SFPixelsPerFoot
            StartY = (bmTopFeet - Ay1(I)) * SFPixelsPerFoot
            StopX = (Bz1(I) - bmLeftFeet) * SFPixelsPerFoot
            StopY = (bmTopFeet - By1(I)) * SFPixelsPerFoot
            g.DrawLine(PlotPen, _
                CSng(StartX), CSng(StartY), CSng(StopX), CSng(StopY))
        Case "XZ"
            StartX = (Ax1(I) - bmLeftFeet) * SFPixelsPerFoot
            StartY = (bmTopFeet - Az1(I)) * SFPixelsPerFoot
            StopX = (Bx1(I) - bmLeftFeet) * SFPixelsPerFoot
            StopY = (bmTopFeet - Bz1(I)) * SFPixelsPerFoot
            g.DrawLine(PlotPen, _
                CSng(StartX), CSng(StartY), CSng(StopX), CSng(StopY))
    End Select
Next I
' Draw the horizontal axes
Select Case Plane
    Case "XY"
        StartX = (0 - bmLeftFeet) * SFPixelsPerFoot
        StartY = (bmTopFeet + 0) * SFPixelsPerFoot
        StopX = ((xMax - bmLeftFeet) + (0.05 * MaxDistance)) * SFPixelsPerFoot
        StopY = (bmTopFeet + 0) * SFPixelsPerFoot
        g.DrawLine(AxisPen, _
            CSng(StartX), CSng(StartY), CSng(StopX), CSng(StopY))
    Case "YZ"
        StartX = ((zMin - bmLeftFeet) - (0.05 * MaxDistance)) * SFPixelsPerFoot
        StartY = (bmTopFeet - 0) * SFPixelsPerFoot
        StopX = ((zMax - bmLeftFeet) + (0.05 * MaxDistance)) * SFPixelsPerFoot
        StopY = (bmTopFeet - 0) * SFPixelsPerFoot
        g.DrawLine(AxisPen, _
            CSng(StartX), CSng(StartY), CSng(StopX), CSng(StopY))
    Case "XZ"
        StartX = (0 - bmLeftFeet) * SFPixelsPerFoot
        StartY = (bmTopFeet - 0) * SFPixelsPerFoot
        StopX = ((xMax - bmLeftFeet) + (0.05 * MaxDistance)) * SFPixelsPerFoot
        StopY = (bmTopFeet - 0) * SFPixelsPerFoot
        g.DrawLine(AxisPen, _
            CSng(StartX), CSng(StartY), CSng(StopX), CSng(StopY))
End Select
' Draw the vertical axes
```

```
        Select Case Plane
            Case "XY"
                StartX = (0 - bmLeftFeet) * SFPixelsPerFoot
                StartY = ((bmTopFeet - yMin) + (0.05 * MaxDistance)) * SFPixelsPerFoot
                StopX = (0 - bmLeftFeet) * SFPixelsPerFoot
                StopY = ((bmTopFeet - 0) - (0.05 * MaxDistance)) * SFPixelsPerFoot
                g.DrawLine(AxisPen, _
                    CSng(StartX), CSng(StartY), CSng(StopX), CSng(StopY))
            Case "YZ"
                StartX = (0 - bmLeftFeet) * SFPixelsPerFoot
                StartY = ((bmTopFeet - yMin) + (0.05 * MaxDistance)) * SFPixelsPerFoot
                StopX = (0 - bmLeftFeet) * SFPixelsPerFoot
                StopY = ((bmTopFeet - 0) - (0.05 * MaxDistance)) * SFPixelsPerFoot
                g.DrawLine(AxisPen, _
                    CSng(StartX), CSng(StartY), CSng(StopX), CSng(StopY))
            Case "XZ"
                StartX = (0 - bmLeftFeet) * SFPixelsPerFoot
                StartY = ((bmTopFeet - zMin) + (0.05 * MaxDistance)) * SFPixelsPerFoot
                StopX = (0 - bmLeftFeet) * SFPixelsPerFoot
                StopY = ((bmTopFeet - zMax) - (0.05 * MaxDistance)) * SFPixelsPerFoot
                g.DrawLine(AxisPen, _
                    CSng(StartX), CSng(StartY), CSng(StopX), CSng(StopY))
        End Select
        ' Dispose of the graphics object
        g.Dispose()
    End Sub

End Module


Option Strict On
Option Explicit On

' There are 11 cross-sections through the blade, starting with the top of the blade and
'   ending with the tip.  The top of the blade is located at D = 6.5 feet and the tip
'   tip is located at D = 11.5 feet.  The spacing is 0.5 feet.  Each point is located at
'   the origin in the corresponding 6-frame of reference.
' There are 5 specified times during a half-stroke, from one extreme of the sweep to the
'   other.  Assume a maximum deflection of 12.4 degrees.
' Vector LocationStart1(11, 5, 3) is the location of one of the 11 points at one of the 5
'   specified times, where the third dimension holds the x, y and z co-ordinates in the
'   1-frame of reference.
' Vector LocationStop1(11, 5, 3) is the location of one of the 11 points one millisecond
'   after the 5 specified times, also expressed in the 1-frame of reference.
' Vectors LocationStart5(11, 5, 3) and LocationStop5(11, 5, 3) are the corresponding
'   location vectors in the 5-frame of reference.
' Vector RelSpeed1(11, 5, 3) is the relative speed expressed in the 1-frame of reference.
' Vector RelSpeed5(11, 5, 3) is the relative speed expressed in the 5-frame of reference.

Public Module AnglesOfAttack

    '////////////////
    '// Data entry //
    '////////////////
    Private Dstart As Double = 6.5
    Private delD As Double = 0.5
    Private D As Double
    Private TStart As Double = 0
```

```
Private delT As Double = 0.25
Private T As Double
Private ThetaDegMax As Double = 12.4
Private ThetaDeg As Double
Private PsiDeg As Double = 40
Private PhiDeg As Double = -45
Private cosTheta, sinTheta As Double
Private cosPsi, sinPsi As Double
Private cosPhi, sinPhi As Double
Private V As Double = 2.35
Private X5, Y5, Z5 As Double
Private X4, Y4, Z4 As Double
Private X3, Y3, Z3 As Double
Private X2, Y2, Z2 As Double
Private X1, Y1, Z1 As Double
Private LocationStart1(11, 5, 3) As Double
Private LocationStop1(11, 5, 3) As Double
Private LocationStart5(11, 5, 3) As Double
Private LocationStop5(11, 5, 3) As Double
Private RelSpeed1(11, 5, 3) As Double
Private RelSpeed5(11, 5, 3) As Double

'//////////////////////
'// Plot parameters //
'//////////////////////
Private BeginPoint(58, 3) As Double
Private EndPoint(58, 3) As Double
Private ViewStart(58, 2) As Double
Private ViewStop(58, 2) As Double
Private Sqrt3 As Double = Math.Sqrt(3)
Private PlotPenT1 As New Drawing.Pen(Color.Red, 2)
Private PlotPenT2 As New Drawing.Pen(Color.Green, 2)
Private PlotPenT3 As New Drawing.Pen(Color.Blue, 2)
Private PlotPenT4 As New Drawing.Pen(Color.Orange, 2)
Private PlotPenT5 As New Drawing.Pen(Color.Violet, 2)
Private AxisPen As New Drawing.Pen(Color.Black, 2)

Public Sub CalculateAnglesOfAttack()
    '
    ' PartA: Calculate the locations at the 5 specified times.
    ' Loop through 11 cross-sections.
    For Iblade As Int32 = 1 To 11 Step 1
        D = Dstart + ((Iblade - 1) * delD)
        ' Loop through 5 specified times during a half-stroke.
        For Itime As Int32 = 1 To 5 Step 1
            ' Calculate the exact time.
            T = TStart + ((Itime - 1) * delT)
            ' Calculate the sweep angle.
            ThetaDeg = ThetaDegMax - (2 * ThetaDegMax * T)
            ' Calculate the trigonometric functions.
            cosTheta = Math.Cos(ThetaDeg * Math.PI / 180)
            sinTheta = Math.Sin(ThetaDeg * Math.PI / 180)
            cosPsi = Math.Cos(PsiDeg * Math.PI / 180)
            sinPsi = Math.Sin(PsiDeg * Math.PI / 180)
            cosPhi = Math.Cos(PhiDeg * Math.PI / 180)
            sinPhi = Math.Sin(PhiDeg * Math.PI / 180)
            ' Set the co-ordinates of the point in the 5-frame of reference.
            LocationStart5(Iblade, Itime, 1) = -D
```

```
        LocationStart5(Iblade, Itime, 2) = 0
        LocationStart5(Iblade, Itime, 3) = 0
        X5 = LocationStart5(Iblade, Itime, 1)
        Y5 = LocationStart5(Iblade, Itime, 2)
        Z5 = LocationStart5(Iblade, Itime, 3)
        ' Transform from frame 5 to frame 4
        X4 = X5
        Y4 = (cosPhi * Y5) + (-sinPhi * Z5)
        Z4 = (sinPhi * Y5) + (cosPhi * Z5)
        ' Transform from frame 4 to frame 3
        X3 = (cosTheta * X4) + (sinTheta * Z4)
        Y3 = Y4
        Z3 = (-sinTheta * X4) + (cosTheta * Z4)
        ' Transform from frame 3 to frame 2
        X2 = (cosPsi * X3) + (-sinPsi * Y3)
        Y2 = (sinPsi * X3) + (cosPsi * Y3)
        Z2 = Z3
        ' Transform from frame 2 to frame 1
        X1 = X2 + (V * T)
        Y1 = Y2
        Z1 = Z2
        ' Save the co-ordinates in the 1-frame of reference.
        LocationStart1(Iblade, Itime, 1) = X1
        LocationStart1(Iblade, Itime, 2) = Y1
        LocationStart1(Iblade, Itime, 3) = Z1
    Next Itime
Next Iblade
'
' PartB: Calculate the locations one millisecond after the 5 specified times.
For Iblade As Int32 = 1 To 11 Step 1
    D = Dstart + ((Iblade - 1) * delD)
    ' Loop through 5 specified times during a half-stroke.
    For Itime As Int32 = 1 To 5 Step 1
        ' Calculate the exact time.
        T = TStart + ((Itime - 1) * delT) + 0.001
        ' Calculate the sweep angle.
        ThetaDeg = ThetaDegMax - (2 * ThetaDegMax * T)
        ' Calculate the trigonometric functions.
        cosTheta = Math.Cos(ThetaDeg * Math.PI / 180)
        sinTheta = Math.Sin(ThetaDeg * Math.PI / 180)
        cosPsi = Math.Cos(PsiDeg * Math.PI / 180)
        sinPsi = Math.Sin(PsiDeg * Math.PI / 180)
        cosPhi = Math.Cos(PhiDeg * Math.PI / 180)
        sinPhi = Math.Sin(PhiDeg * Math.PI / 180)
        ' Set the co-ordinates of the point in the 5-frame of reference.
        LocationStop5(Iblade, Itime, 1) = -D
        LocationStop5(Iblade, Itime, 2) = 0
        LocationStop5(Iblade, Itime, 3) = 0
        X5 = LocationStop5(Iblade, Itime, 1)
        Y5 = LocationStop5(Iblade, Itime, 2)
        Z5 = LocationStop5(Iblade, Itime, 3)
        ' Transform from frame 5 to frame 4
        X4 = X5
        Y4 = (cosPhi * Y5) + (-sinPhi * Z5)
        Z4 = (sinPhi * Y5) + (cosPhi * Z5)
        ' Transform from frame 4 to frame 3
        X3 = (cosTheta * X4) + (sinTheta * Z4)
        Y3 = Y4
```

```
        Z3 = (-sinTheta * X4) + (cosTheta * Z4)
        ' Transform from frame 3 to frame 2
        X2 = (cosPsi * X3) + (-sinPsi * Y3)
        Y2 = (sinPsi * X3) + (cosPsi * Y3)
        Z2 = Z3
        ' Transform from frame 2 to frame 1
        X1 = X2 + (V * T)
        Y1 = Y2
        Z1 = Z2
        ' Save the co-ordinates in the 1-frame of reference.
        LocationStop1(Iblade, Itime, 1) = X1
        LocationStop1(Iblade, Itime, 2) = Y1
        LocationStop1(Iblade, Itime, 3) = Z1
    Next Itime
Next Iblade
'
' Part C: Calculate the relative speed.
For Iblade As Int32 = 1 To 11 Step 1
    For Itime As Int32 = 1 To 5 Step 1
        RelSpeed1(Iblade, Itime, 1) = (LocationStop1(Iblade, Itime, 1) - _
            LocationStart1(Iblade, Itime, 1)) / 0.001
        RelSpeed1(Iblade, Itime, 2) = (LocationStop1(Iblade, Itime, 2) - _
            LocationStart1(Iblade, Itime, 2)) / 0.001
        RelSpeed1(Iblade, Itime, 3) = (LocationStop1(Iblade, Itime, 3) - _
            LocationStart1(Iblade, Itime, 3)) / 0.001
        ' Calculate the exact time.
        T = TStart + ((Itime - 1) * delT)
        ' Calculate the sweep angle.
        ThetaDeg = ThetaDegMax - (2 * ThetaDegMax * T)
        ' Calculate the trigonometric functions.
        cosTheta = Math.Cos(ThetaDeg * Math.PI / 180)
        sinTheta = Math.Sin(ThetaDeg * Math.PI / 180)
        cosPsi = Math.Cos(PsiDeg * Math.PI / 180)
        sinPsi = Math.Sin(PsiDeg * Math.PI / 180)
        cosPhi = Math.Cos(PhiDeg * Math.PI / 180)
        sinPhi = Math.Sin(PhiDeg * Math.PI / 180)
        ' Rotate the relative speed back to the 5-frame of reference.
        ' Transform from frame 1 to frame 2
        X2 = RelSpeed1(Iblade, Itime, 1)
        Y2 = RelSpeed1(Iblade, Itime, 2)
        Z2 = RelSpeed1(Iblade, Itime, 3)
        ' Transform from frame 2 to frame 3
        X3 = (cosPsi * X2) + (sinPsi * Y2)
        Y3 = (-sinPsi * X2) + (cosPsi * Y2)
        Z3 = Z2
        ' Transform from frame 3 to frame 4
        X4 = (cosTheta * X3) + (-sinTheta * Z3)
        Y4 = Y3
        Z4 = (sinTheta * X3) + (cosTheta * Z3)
        ' Transform from frame 4 to frame 5
        X5 = X4
        Y5 = (cosPhi * Y4) + (sinPhi * Z4)
        Z5 = (-sinPhi * Y4) + (cosPhi * Z4)
        ' Save the relative speed in the 5-frame of reference.
        RelSpeed5(Iblade, Itime, 1) = X5
        RelSpeed5(Iblade, Itime, 2) = Y5
        RelSpeed5(Iblade, Itime, 3) = Z5
    Next Itime
```

```
    Next Iblade

    Form1.TextArea.Text = ""
    For Iblade As Int32 = 1 To 11 Step 5
        For Itime As Int32 = 1 To 5 Step 4
            Dim X5 As Double = RelSpeed5(Iblade, Itime, 1)
            Dim Y5 As Double = RelSpeed5(Iblade, Itime, 2)
            Dim Z5 As Double = RelSpeed5(Iblade, Itime, 3)
            Dim SigmaRad As Double = Math.Atan(-X5 / Z5)
            Dim SigmaDeg As Double = SigmaRad * 180 / Math.PI
            Dim Speed As Double = Math.Sqrt((X5 * X5) + (Y5 * Y5) + (Z5 * Z5))
            Dim ProjLen As Double = Math.Sqrt((X5 * X5) + (Z5 * Z5))
            Dim AlphaRad As Double = Math.Acos(ProjLen / Speed)
            Dim AlphaDeg As Double = AlphaRad * 180 / Math.PI
            Form1.TextArea.Text = Form1.TextArea.Text & _
                "Iblade=" & Trim(Str(Iblade)) & _
                "  Itime=" & Trim(Str(Itime)) & vbCrLf & _
                "  SpdX=" & Trim(Str(X5)) & _
                "  SpdY=" & Trim(Str(Y5)) & _
                "  SpdZ=" & Trim(Str(Z5)) & _
                "  Sigma=" & Trim(Str(SigmaDeg)) & _
                "  Alpha=" & Trim(Str(AlphaDeg)) & _
                "  Speed=" & Trim(Str(Speed)) & vbCrLf
        Next Itime
    Next Iblade

End Sub

Public Sub RenderAnglesOfAttackIn1Frame( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs, _
    ByRef PlotBitmap As Bitmap)
    ' This subroutine plots a set of 55 line segments whose starting locations, in
    ' feet, and effective lengths, in feet per second, are given in the two vectors
    ' LocationStart1() and RelSpeed1(), respectively.  The canvas is orthogonal with
    ' the Y-axis pointing up, the X-axis extending towards the lower right at a
    ' 30-degree angle and the Z-axis extending towards the lower left at a 30-degree
    ' angle.  Separate scale factors are applied to the locations and lengths to
    ' enable easy adjustment of the figure.  Note that the scaling factors are set
    ' manually.
    Dim SFPixelsPerFoot As Double = 75
    Dim SFPixelsPerFPS As Double = 10
    ' Transfer the data into 55 consecutive beginning and ending points.
    For Iblade As Int32 = 1 To 11 Step 1
        For Itime As Int32 = 1 To 5 Step 1
            Dim IndexInVector As Int32
            IndexInVector = ((Iblade - 1) * 5) + Itime
            BeginPoint(IndexInVector, 1) = _
                LocationStart1(Iblade, Itime, 1) * SFPixelsPerFoot
            BeginPoint(IndexInVector, 2) = _
                LocationStart1(Iblade, Itime, 2) * SFPixelsPerFoot
            BeginPoint(IndexInVector, 3) = _
                LocationStart1(Iblade, Itime, 3) * SFPixelsPerFoot
            EndPoint(IndexInVector, 1) = BeginPoint(IndexInVector, 1) + _
                (RelSpeed1(Iblade, Itime, 1) * SFPixelsPerFPS)
            EndPoint(IndexInVector, 2) = BeginPoint(IndexInVector, 2) + _
                (RelSpeed1(Iblade, Itime, 2) * SFPixelsPerFPS)
            EndPoint(IndexInVector, 3) = BeginPoint(IndexInVector, 3) + _
```

```
            (RelSpeed1(Iblade, Itime, 3) * SFPixelsPerFPS)
    Next Itime
Next Iblade
' Add a line for the X-axis.
BeginPoint(56, 1) = -7 * SFPixelsPerFoot
BeginPoint(56, 2) = 0
BeginPoint(56, 3) = 0
EndPoint(56, 1) = 0.5 * SFPixelsPerFoot
EndPoint(56, 2) = 0
EndPoint(56, 3) = 0
' Add a line for the Y-axis.
BeginPoint(57, 1) = 0
BeginPoint(57, 2) = -7 * SFPixelsPerFoot
BeginPoint(57, 3) = 0
EndPoint(57, 1) = 0
EndPoint(57, 2) = 0.5 * SFPixelsPerFoot
EndPoint(57, 3) = 0
' Add a line for the Z-axis
BeginPoint(58, 1) = 0
BeginPoint(58, 2) = 0
BeginPoint(58, 3) = -2 * SFPixelsPerFoot
EndPoint(58, 1) = 0
EndPoint(58, 2) = 0
EndPoint(58, 3) = 2 * SFPixelsPerFoot
' Convert the data to a two-dimensional framework.  The vector ViewStart(55,2)
' contains the horizontal and vertical co-ordinates of the starts of the 55 line
' segments.  Vector ViewStop(55,2) are the co-ordinates of the ends of the
' corresponding line segments.  The dimensions are expressed in pixels with
' respect to the (0, 0, 0) origin.
For I As Int32 = 1 To 58 Step 1
    ViewStart(I, 1) = _
        (BeginPoint(I, 1) * Sqrt3 / 2) + _
        (-BeginPoint(I, 3) * Sqrt3 / 2)
    ViewStart(I, 2) = _
        (-BeginPoint(I, 1) / 2) + _
        BeginPoint(I, 2) + _
        (-BeginPoint(I, 3) / 2)
    ViewStop(I, 1) = _
        (EndPoint(I, 1) * Sqrt3 / 2) + _
        (-EndPoint(I, 3) * Sqrt3 / 2)
    ViewStop(I, 2) = _
        (-EndPoint(I, 1) / 2) + _
        EndPoint(I, 2) + _
        (-EndPoint(I, 3) / 2)
Next I
' Translate the origin to the center of the PlotArea.
For I As Int32 = 1 To 58 Step 1
    ViewStart(I, 1) = 800 + ViewStart(I, 1)
    ViewStart(I, 2) = 250 - ViewStart(I, 2)
    ViewStop(I, 1) = 800 + ViewStop(I, 1)
    ViewStop(I, 2) = 250 - ViewStop(I, 2)
Next I
' Define the graphics object
Dim g As Graphics = Graphics.FromImage(PlotBitmap)
' Draw the segments one-by-one starting.  A small dot is placed at the starting
' location so the direction of motion can be better understood.  The five
' across a sweep are coloured in the order: red, green, blue, orange, violet.
' Note that the last three segments are axes and should be rendered using the
```

```
    ' appropriate colour.
    For Iblade As Int32 = 1 To 11 Step 1
        For Itime As Int32 = 1 To 5 Step 1
            Dim J As Int32
            J = ((Iblade - 1) * 5) + Itime
            Select Case Itime
                Case 1
                    g.DrawLine(PlotPenT1, _
                        CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                        CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
                Case 2
                    g.DrawLine(PlotPenT2, _
                        CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                        CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
                Case 3
                    g.DrawLine(PlotPenT3, _
                        CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                        CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
                Case 4
                    g.DrawLine(PlotPenT4, _
                        CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                        CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
                Case 5
                    g.DrawLine(PlotPenT5, _
                        CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                        CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
            End Select
            g.FillEllipse(Brushes.Black, _
                CSng(ViewStart(J, 1) - 3), CSng(ViewStart(J, 2) - 3), 6, 6)
        Next Itime
    Next Iblade
    For I As Int32 = 56 To 58 Step 1
        g.DrawLine(AxisPen, _
            CSng(ViewStart(I, 1)), CSng(ViewStart(I, 2)), _
            CSng(ViewStop(I, 1)), CSng(ViewStop(I, 2)))
    Next I
    ' Dispose of the graphics object
    g.Dispose()
End Sub

Public Sub RenderAnglesOfAttackIn6Frame( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs, _
    ByRef PlotBitmap As Bitmap)
    ' This subroutine plots a set of 55 line segments whose starting locations, in
    ' feet, and effective lengths, in feet per second, are given in the two vectors
    ' LocationStart5() and RelSpeed5(), respectively.  The canvas is orthogonal with
    ' the Y-axis pointing up, the X-axis extending towards the lower right at a
    ' 30-degree angle and the Z-axis extending towards the lower left at a 30-degree
    ' angle.  Separate scale factors are applied to the locations and lengths to
    ' enable easy adjustment of the figure.  Note that the scaling factors are set
    ' manually.
    Dim SFPixelsPerFoot As Double = 100
    Dim SFPixelsPerFPS As Double = 50
    ' Transfer the data into 55 consecutive beginning and ending points.
    For Iblade As Int32 = 1 To 11 Step 1
        For Itime As Int32 = 1 To 5 Step 1
            Dim IndexInVector As Int32
```

```
        IndexInVector = ((Iblade - 1) * 5) + Itime
        D = -((Iblade - 1) * delD)
        BeginPoint(IndexInVector, 1) = D * SFPixelsPerFoot
        BeginPoint(IndexInVector, 2) = 0
        BeginPoint(IndexInVector, 3) = 0
        EndPoint(IndexInVector, 1) = BeginPoint(IndexInVector, 1) + _
            (RelSpeed5(Iblade, Itime, 1) * SFPixelsPerFPS)
        EndPoint(IndexInVector, 2) = BeginPoint(IndexInVector, 2) + _
            (RelSpeed5(Iblade, Itime, 2) * SFPixelsPerFPS)
        EndPoint(IndexInVector, 3) = BeginPoint(IndexInVector, 3) + _
            (RelSpeed5(Iblade, Itime, 3) * SFPixelsPerFPS)
    Next Itime
Next Iblade
' Add a line for the X-axis.
BeginPoint(56, 1) = -6 * SFPixelsPerFoot
BeginPoint(56, 2) = 0
BeginPoint(56, 3) = 0
EndPoint(56, 1) = 0.5 * SFPixelsPerFoot
EndPoint(56, 2) = 0
EndPoint(56, 3) = 0
' Add a line for the Y-axis.
BeginPoint(57, 1) = 0
BeginPoint(57, 2) = -0.5 * SFPixelsPerFoot
BeginPoint(57, 3) = 0
EndPoint(57, 1) = 0
EndPoint(57, 2) = 0.5 * SFPixelsPerFoot
EndPoint(57, 3) = 0
' Add a line for the Z-axis
BeginPoint(58, 1) = 0
BeginPoint(58, 2) = 0
BeginPoint(58, 3) = -2 * SFPixelsPerFoot
EndPoint(58, 1) = 0
EndPoint(58, 2) = 0
EndPoint(58, 3) = 2 * SFPixelsPerFoot
' Convert the data to a two-dimensional framework.  The vector ViewStart(55,2)
' contains the horizontal and vertical co-ordinates of the starts of the 55 line
' segments.  Vector ViewStop(55,2) are the co-ordinates of the ends of the
' corresponding line segments.  The dimensions are expressed in pixels with
' respect to the (0, 0, 0) origin.
For I As Int32 = 1 To 58 Step 1
    ViewStart(I, 1) = _
        (BeginPoint(I, 1) * Sqrt3 / 2) + _
        (-BeginPoint(I, 3) * Sqrt3 / 2)
    ViewStart(I, 2) = _
        (-BeginPoint(I, 1) / 2) + _
        BeginPoint(I, 2) + _
        (-BeginPoint(I, 3) / 2)
    ViewStop(I, 1) = _
        (EndPoint(I, 1) * Sqrt3 / 2) + _
        (-EndPoint(I, 3) * Sqrt3 / 2)
    ViewStop(I, 2) = _
        (-EndPoint(I, 1) / 2) + _
        EndPoint(I, 2) + _
        (-EndPoint(I, 3) / 2)
Next I
' Translate the origin to the center of the PlotArea.
For I As Int32 = 1 To 58 Step 1
    ViewStart(I, 1) = 600 + ViewStart(I, 1)
```

```
            ViewStart(I, 2) = 500 - ViewStart(I, 2)
            ViewStop(I, 1) = 600 + ViewStop(I, 1)
            ViewStop(I, 2) = 500 - ViewStop(I, 2)
        Next I
        ' Define the graphics object
        Dim g As Graphics = Graphics.FromImage(PlotBitmap)
        ' Draw the segments one-by-one starting.  A small dot is placed at the starting
        ' location so the direction of motion can be better understood.  The five
        ' across a sweep are coloured in the order: red, green, blue, orange, violet.
        ' Note that the last three segments are axes and should be rendered using the
        ' appropriate colour.
        For Iblade As Int32 = 1 To 11 Step 1
            For Itime As Int32 = 1 To 5 Step 1
                Dim J As Int32
                J = ((Iblade - 1) * 5) + Itime
                Select Case Itime
                    Case 1
                        g.DrawLine(PlotPenT1, _
                            CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                            CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
                    Case 2
                        g.DrawLine(PlotPenT2, _
                            CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                            CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
                    Case 3
                        g.DrawLine(PlotPenT3, _
                            CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                            CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
                    Case 4
                        g.DrawLine(PlotPenT4, _
                            CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                            CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
                    Case 5
                        g.DrawLine(PlotPenT5, _
                            CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                            CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
                End Select
                g.FillEllipse(Brushes.Black, _
                    CSng(ViewStart(J, 1) - 3), CSng(ViewStart(J, 2) - 3), 6, 6)
            Next Itime
        Next Iblade
        For I As Int32 = 56 To 58 Step 1
            g.DrawLine(AxisPen, _
                CSng(ViewStart(I, 1)), CSng(ViewStart(I, 2)), _
                CSng(ViewStop(I, 1)), CSng(ViewStop(I, 2)))
        Next I
        ' Dispose of the graphics object
        g.Dispose()
    End Sub

    Public Sub RenderAnglesOfAttackIn6XZFrame( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs, _
        ByRef PlotBitmap As Bitmap)
        ' This subroutine plots the relative speed in the 6-frame of reference, but only
        ' the components of the speed in the X-Z plane.  The X-axis points to the right
        ' and the Z-axis points straight down.
        Dim SFPixelsPerFoot As Double = 100
```

```
Dim SFPixelsPerFPS As Double = 50
' Transfer the data into 55 consecutive beginning and ending points.
For Iblade As Int32 = 1 To 11 Step 1
    For Itime As Int32 = 1 To 5 Step 1
        Dim IndexInVector As Int32
        IndexInVector = ((Iblade - 1) * 5) + Itime
        D = -((Iblade - 1) * delD)
        BeginPoint(IndexInVector, 1) = D * SFPixelsPerFoot
        BeginPoint(IndexInVector, 3) = 0
        EndPoint(IndexInVector, 1) = BeginPoint(IndexInVector, 1) + _
            (RelSpeed5(Iblade, Itime, 1) * SFPixelsPerFPS)
        EndPoint(IndexInVector, 3) = BeginPoint(IndexInVector, 3) + _
            (RelSpeed5(Iblade, Itime, 3) * SFPixelsPerFPS)
    Next Itime
Next Iblade
' Add a line for the X-axis.
BeginPoint(56, 1) = -6 * SFPixelsPerFoot
BeginPoint(56, 3) = 0
EndPoint(56, 1) = 0.5 * SFPixelsPerFoot
EndPoint(56, 3) = 0
' Add a line for the Z-axis
BeginPoint(57, 1) = 0
BeginPoint(57, 3) = -2 * SFPixelsPerFoot
EndPoint(57, 1) = 0
EndPoint(57, 3) = 2 * SFPixelsPerFoot
' Convert the data to a two-dimensional framework.  The vector ViewStart(55,2)
' contains the horizontal and vertical co-ordinates of the starts of the 55 line
' segments.  Vector ViewStop(55,2) are the co-ordinates of the ends of the
' corresponding line segments.  The dimensions are expressed in pixels with
' respect to the (0, 0, 0) origin.
For I As Int32 = 1 To 57 Step 1
    ViewStart(I, 1) = BeginPoint(I, 1)
    ViewStart(I, 2) = -BeginPoint(I, 3)
    ViewStop(I, 1) = EndPoint(I, 1)
    ViewStop(I, 2) = -EndPoint(I, 3)
Next I
' Translate the origin to the center of the PlotArea.
For I As Int32 = 1 To 57 Step 1
    ViewStart(I, 1) = 700 + ViewStart(I, 1)
    ViewStart(I, 2) = 400 - ViewStart(I, 2)
    ViewStop(I, 1) = 700 + ViewStop(I, 1)
    ViewStop(I, 2) = 400 - ViewStop(I, 2)
Next I
' Define the graphics object
Dim g As Graphics = Graphics.FromImage(PlotBitmap)
' Draw the segments one-by-one starting.  Note that the last two segments
' are axes and should be rendered using the appropriate colour.
For Iblade As Int32 = 1 To 11 Step 1
    For Itime As Int32 = 1 To 5 Step 1
        Dim J As Int32
        J = ((Iblade - 1) * 5) + Itime
        Select Case Itime
            Case 1
                g.DrawLine(PlotPenT1, _
                    CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                    CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
            Case 2
                g.DrawLine(PlotPenT2, _
```

```
                        CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                        CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
                Case 3
                    g.DrawLine(PlotPenT3, _
                        CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                        CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
                Case 4
                    g.DrawLine(PlotPenT4, _
                        CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                        CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
                Case 5
                    g.DrawLine(PlotPenT5, _
                        CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                        CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
            End Select
            g.FillEllipse(Brushes.Black, _
                CSng(ViewStart(J, 1) - 3), CSng(ViewStart(J, 2) - 3), 6, 6)
        Next Itime
    Next Iblade
    For I As Int32 = 56 To 57 Step 1
        g.DrawLine(AxisPen, _
            CSng(ViewStart(I, 1)), CSng(ViewStart(I, 2)), _
            CSng(ViewStop(I, 1)), CSng(ViewStop(I, 2)))
    Next I
    ' Dispose of the graphics object
    g.Dispose()
End Sub

Public Sub RenderAnglesOfAttackIn6YZFrame( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs, _
    ByRef PlotBitmap As Bitmap)
    ' This subroutine plots the relative speed in the 6-frame of reference, but only
    ' the components of the speed in the Y-Z plane.  The Y-axis points straight up
    ' and the Z-axis points to the *LEFT**.
    Dim SFPixelsPerFoot As Double = 100
    Dim SFPixelsPerFPS As Double = 100
    ' Transfer the data into 55 consecutive beginning and ending points.
    For Iblade As Int32 = 1 To 11 Step 1
        For Itime As Int32 = 1 To 5 Step 1
            Dim IndexInVector As Int32
            IndexInVector = ((Iblade - 1) * 5) + Itime
            BeginPoint(IndexInVector, 2) = 0
            BeginPoint(IndexInVector, 3) = 0
            EndPoint(IndexInVector, 2) = BeginPoint(IndexInVector, 2) + _
                (RelSpeed5(Iblade, Itime, 2) * SFPixelsPerFPS)
            EndPoint(IndexInVector, 3) = BeginPoint(IndexInVector, 3) + _
                (RelSpeed5(Iblade, Itime, 3) * SFPixelsPerFPS)
        Next Itime
    Next Iblade
    ' Add a line for the Y-axis.
    BeginPoint(56, 2) = -0.5 * SFPixelsPerFoot
    BeginPoint(56, 3) = 0
    EndPoint(56, 2) = 0.5 * SFPixelsPerFoot
    EndPoint(56, 3) = 0
    ' Add a line for the Z-axis
    BeginPoint(57, 2) = 0
    BeginPoint(57, 3) = -2 * SFPixelsPerFoot
```

```
EndPoint(57, 2) = 0
EndPoint(57, 3) = 2 * SFPixelsPerFoot
' Convert the data to a two-dimensional framework.  The vector ViewStart(55,2)
' contains the horizontal and vertical co-ordinates of the starts of the 55 line
' segments.  Vector ViewStop(55,2) are the co-ordinates of the ends of the
' corresponding line segments.  The dimensions are expressed in pixels with
' respect to the (0, 0, 0) origin.
For I As Int32 = 1 To 57 Step 1
    ViewStart(I, 1) = -BeginPoint(I, 3)
    ViewStart(I, 2) = BeginPoint(I, 2)
    ViewStop(I, 1) = -EndPoint(I, 3)
    ViewStop(I, 2) = EndPoint(I, 2)
Next I
' Translate the origin to the center of the PlotArea.
For I As Int32 = 1 To 57 Step 1
    ViewStart(I, 1) = 400 + ViewStart(I, 1)
    ViewStart(I, 2) = 400 - ViewStart(I, 2)
    ViewStop(I, 1) = 400 + ViewStop(I, 1)
    ViewStop(I, 2) = 400 - ViewStop(I, 2)
Next I
' Define the graphics object
Dim g As Graphics = Graphics.FromImage(PlotBitmap)
' Draw the segments one-by-one starting.  Note that the last two segments
' are axes and should be rendered using the appropriate colour.
For Iblade As Int32 = 1 To 11 Step 10
    For Itime As Int32 = 1 To 5 Step 1
        Dim J As Int32
        J = ((Iblade - 1) * 5) + Itime
        Select Case Itime
            Case 1
                g.DrawLine(PlotPenT1, _
                    CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                    CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
            Case 2
                g.DrawLine(PlotPenT2, _
                    CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                    CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
            Case 3
                g.DrawLine(PlotPenT3, _
                    CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                    CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
            Case 4
                g.DrawLine(PlotPenT4, _
                    CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                    CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
            Case 5
                g.DrawLine(PlotPenT5, _
                    CSng(ViewStart(J, 1)), CSng(ViewStart(J, 2)), _
                    CSng(ViewStop(J, 1)), CSng(ViewStop(J, 2)))
        End Select
        g.FillEllipse(Brushes.Black, _
            CSng(ViewStart(J, 1) - 3), CSng(ViewStart(J, 2) - 3), 6, 6)
    Next Itime
Next Iblade
For I As Int32 = 56 To 57 Step 1
    g.DrawLine(AxisPen, _
        CSng(ViewStart(I, 1)), CSng(ViewStart(I, 2)), _
        CSng(ViewStop(I, 1)), CSng(ViewStop(I, 2)))
```

```
        Next I
        ' Dispose of the graphics object
        g.Dispose()
    End Sub

End Module
```