Intro to 3D Graphics Programming:

What You Need To Know First:

I'm assuming you have a knowledge of basic algebra, i.e. early highschool level math. If you've had some Trigonometry already, that will helpout, but I'll try to cover
the basics of what you need from that as well.But, more important than any one particular requirement, you have to likemath, or at least a willingness to like it.

MATH IS ONLY BORING IF YOU DON'T HAVE A USE FOR IT.

Basically, 3D coding is not really about programming... it's almostentirely about math. The amount of time you spend thinking about what you'redoing is generally a
lot longer than the time you actually spend poundingthe keys writing the code.

If you don't think you like math, and you're a coder, chances are youprobably _DO_ like math and just don't know it. :-) I'm serious... a lotof times when you hate a
subject in school, you don't really hate it, youjust think you hate it because you hate the way it's taught. Once you getdown to finding a purpose for all that "useless"
knowledge....it becomes a heck of a lot of fun. So if you like math, and you like codinggraphics, you've come to the right place. If you like coding graphics butdon't
like math... give it a chance; you may find out something new aboutyourself. :-)

What You Don't Need to Know First

Okay, generally speaking, the chain of math courses taught in high schooland college (or their European equivalents, which I can't really speakfor but I can take a
guess) goes something like this:

1. Algebra 2. Geometry 3. Analytical Algebra 4. Trigonometry 5. Pre-Calculus6. Calculus 7. Linear Algebra 8. Vector Geometry 9. Multivariable Calculus10.
Differential Equations 11. Even more complicated issues that I haven'tgotten to yet...

You need to know up to #3 to make use of any of this. #4 would be helpful,although I'll cover a few things here in the beginning to get you going,if you've never
taken Trig before.

Pre-Calc and Calc aren't directly needed for this series, although you'llfind it useful later on in life when you want to make really cool movementsfor your vectors.
And if you know through #7, life is a breeze; you canforget most of this beginning crap. :) Anything after that I'm not goingto cover by itself, but you'll find it helpful if
you know it, as you trymore innovative ideas with your vector code. But still, it's not necessaryat all for the little I'm going to be able to teach you. :)

What Will Be Covered In This Series

Whew, it's a whopper. Here's the series layout as I'm planning it sofar. As you can tell, the first couple articles are for the _very_ beginners,like the kind who know
how to code, but have never even touched anythingwith 3D before. After the first few, it'll get more interesting though,so bear with me (if you have some 3D
experience already, you can forgetreading the first few articles; they won't do you any good).

1. Basic Trig functions, 3D Coordinate to 2D Screen Projection. 2. 2Dand 3D Rotations & Other Transformations. 3. Dot and Cross Products,Backface Removal
and Lightsourcing. 4. Polygon Fills - Flat and Lambert5. Polygon Fills - Gouraud and Phong 6. Polygon Fills - Affine TextureMapping 7. Polygon Fills -

Perspective Texture Mapping, Perspective Correction8. Spherical and Cylindrical Coordinates 9. Polygon Fills - Bump Mapping10. BSP Trees and Spacial
Partitioning 11. Polygon Fills - Reflection Mapping12. Alternative Camera Positioning and Motion 13. Path Management, Linearand Bezier Curve Movement 14.
Basic Inverse Kinematics 15. (Unknown - I'mnot sure how much further I can take it without risking losing the fewtricks I have up my sleeve for my future :)

It's quite likely that the later articles will be split into separatearticles as well (I doubt I can fit both Gouraud and Phong, for example,into one article). So the
numbers are probably going to get higher andhigher. It will probably be around the time of NAID or afterward that theseries ends.

Get ready for a crash course in 3D... :-)

Section One - Basic TrigonometricFunctions

Okay, If you've taken Trig before, skip this section. If you haven't,you're probably thinking that this long title doesn't sound too friendly.Don't worry, it sounds a
whole lot worse than it is.

The main principles of Trigonometry come from the first part of thename, "Trigon", which pretty much means a triangle. Everythingin Trig is based on a few functions
which deal with triangles. There aresix of these functions...

Sine, Cosine, Tangent, Secant, Cosecant, and Cotangent Abbreviated:Sin, Cos, Tan, Sec, Csc, and Cot.

... but in fact, they all boil down to the first two, Sine and Cosine.

Now I assume you're all familiar with functions, like [f(x) = 2x] andso forth (BTW I'll enclose functions in [] on occasion throughout thisdoc). Well the Trig functions
are all functions using angles, for example[f(x) = sin(x)] or even [f(x) = 2 sin(x) - 5 cos(x)], and so on. In thesecases, x is an angle, either in degrees or radians. I'm
going to stickwith degrees for this explanation, since radians are too complex for meto discuss right now.

Okay, so now you're thinking, "Great... Sine and Cosine are functionsusing angles. But what do they _DO_?" Well, here's the rundown.

Get out a piece of paper. Go ahead, I'll wait. Now on that piece ofpaper, draw yourself a pair of X and Y coordinate axes, like you normallywould for a graph.
Make it big, so you give yourself lots of room.

Now draw a circle around the origin. Make it as good a circle as youcan sketch. Now whatever size you drew your circle, assume that's a radiusof 1. So where the
circle hits your X and Y axes, label those points asX=1 on the right, Y=1 on the top, X=-1 on the left, and Y=-1 on the bottom.

Okay, with this circle, we need some angles. Using your good ol' 360-degreesystem, label those four points with their angles, where 0 degrees is facingthe right and
they go counterclockwise. So 0 is at the right, 90 is up,180 is left, and 270 is down.

Now knowing that's how the angles go, draw a line from the origin outto the circle at about, oh, 60 degrees (up and to the right, more tallthan wide). At that point
on the circle, draw a line STRAIGHT DOWN to theX axis. What you should have now is a triangle inside the circle, withone side on the X axis, one going straight
up, and one at a 60 degree angle.

Okay, it's time for a little guessing game. Assuming that the circlehas a radius of 1 like you drew it, how tall is that vertical side of thetriangle? It's almost as tall as the
circle, but not quite... more thanhalf the height of the circle, maybe around 3/4 the height, something likethat.

What was your guess? Probably somewhere between 0.7 and 0.9, I wouldthink. It should look about that tall. Whatever your guess is, write itdown, and keep it
handy... it's going to be very important.

Now do the same thing for the bottom horizontal side of the triangle,the one along the X axis. How long is that from the looks of it? It appearsaround half the width
of the circle, so you'd probably guess around 0.5or so. Anyway, write that one down too.

Okay, So you've got this triangle inside this circle, with the longestside at an angle of 60 degrees and a length of 1 (the radius of the circle),and two sides going
straight vertical and horizontal, with lengths somewherenear the numbers you guessed.

I'm gonna pull out my calculator, which supports Trig functions. Lemmepunch up the numbers....

Sin(60) = 0.8660254 Cos(60) = 0.5000000

Notice any correlation? You should... they're the lengths of the sidesyou were guessing! :-) That, quite simply, is what the Sin and Cos functionsare... if you have a
circle with radius 1, and a line at any angle fromthe origin to the circle, then the Sine of that angle is the Y-component(height) of that line, and the Cosine of the angle
is the X-component (width)of the line.

Other examples? Well now that you know what the Sin and Cos functionsmean, what are the Sin and Cos of 90 degrees? Well let's see... that'sthe line going
straight up from the origin, along the Y axis. The heightis the height of circle, exactly. And the width... well, there is no width,since it's not going left or right
whatsoever.

Sure enough,

Sin(90) = 1.0000000 Cos(90) = 0.0000000

Likewise, the trend continues.....

Angle Sine Cosine 0 0 1 90 1 0 180 0 -1 270 -1 0

Which demonstrates a very important point about these functions

Fact: The Sine and Cosine functions will never go less than -1, andnever greater than 1.

... and that makes sense, considering the circle has a radius of 1.(Terminology: The circle you drew is often called the "unit circle"by math books and teacher-type
people).

Now by looking at the circle, you can see how the height changes lessand less as you approach Sin=1 or Sin=-1, and likewise the width changesless and less as you
approach Cos=1 or Cos=-1. It turns out that the Sinand Cos functions are not linear whatsoever... and because of that, calculatingthem for any given angle is a real
pain in the rear. Sure, you have theoccasional angles like 90 and 180 which have 1s and 0s, or 30 and 60 whichhave one of the components exactly 0.5, but most
of the time you have valuesthat are not the nicest of numbers.

For this reason, you often hear 3D coders referring to "Sine Tables",which are precalculated tables holding the Sin and Cos values for everyangle in their system (in

this example, a 360 degree system. It turns outin coding that a 256 degree system is much more convenient, for reasonsyou can probably guess). These do-it-once
tables turn an otherwise painfulcalculation into a simple memory look-up. :-)

(BTW - earlier I mentioned in addition to Sin and Cos the Tan, Sec,Csc and Cot functions... these can all be created from Sin and Cos by thefollowing: Tan(x) =
Sin(x)/Cos(x) Sec(x) = 1/Cos(x) Csc(x) = 1/Sin(x) Cot(x) = Cos(x)/Sin(x) = 1/Tan(x) Most of those aren't very useful invector coding, with the exception of
Tangent... it will come in handy attimes. Everything else isn't too important for now, but keep it on theback burner for future reference).

Okay, so now you've learned what you need to know from Trig... thatthe Sin and Cos functions can be used to find the X and Y parts of a lineat any given angle.
You just take the length of the line, multiply theSin or Cos (whichever you're looking for, Y or X respectively) by the line'slength, and wham you've got your result.
This will be used EXTENSIVELYin the rest of the doc and in 3D in general, as almost everything dependson these components.

There is certainly a lot more to trigonometry, and entire books havebeen dedicated to the subject. My highly over-simplified version here shouldhopefully be enough
to get your foot in the door. If you want more complexand detailed info, I'd highly suggest a good math textbook covering thesubject (Trig is often grouped in with
pre-calculus texts, incase you havea difficult time finding a separate book). But this should be good enoughfor the time being. :)

All set? Then let's get out of this plane of thought (sorry, very badpun) and get into some three-space nitty gritty, starting with...

Section Two - The 3D Cartesian CoordinateSystem

Okay, you're undoubtedly already used to doing graphics in 2D usingthe X and Y axes; it's what you've been doing in algebra all along. Wellusing X and Y is the
2D standard of the Cartesian coordinate system, andlikewise, we need to add another axis of depth for 3D - The Z axis. Butwhere does this Z axis go?

Well X and Y are perpendicular to each other in the 2D plane. It's thesame as the plane of your screen, for example. Well in 3D, we can get depthby putting the Z
axis perpendicular to that entire plane altogether, goingeither into or out of your screen.

A common term for a line or plane perpendicular to another plane is"orthogonal". That is, we want a Z axis that's orthogonal tothe XY screen plane.

But in which direction? It can either go into or out of the plane, sowhich one is right? Well this is really entirely by convention... the standardway is to use a "right
handed" orientation. What's a right handedorientation? Okay, hold up your right hand in front of your face. Pointyour thumb toward your nose. Now look at your
fingers... they curl counterclockwise.This is the basis behind a right-handed system. When you have the XY plane,the two axes for X and Y are to the right and
going up, respectively. Soif you start your hand at the first axis, X, and curl your right hand'sfingers in the direction of the second axis, Y, your thumb will point
outtoward your nose.

So the Z axis goes out of the screen, toward you, by a right-handedsystem (for the same reason, if you used the YX plane instead of the XYplane, the curl would
be in the opposite direction and Z would go intothe screen instead).

Cheesy ASCII graph....

Y | | | Screen Plane | | -------X / / / Z

The graph above is the way we're going to do it in this doc. All 3DCartesian really is, is just the same coordinate system you're used to,but with the added
dimension so you can locate a point anywhere in spaceand not just on a plane. End of story (You might find later that othercoordinate systems, like spherical and
cylindrical, can help out with someroutines... but I'll go over those in time).

Short section, eh? There's more, but not much. :-) Well, now with the3D system in your grasp, it's time to put it to use, and get some 3D pointsto show up
on-screen.

Section Three - Perspective Projection

Now we've got this nice XYZ spacial coordinate system... but how dowe take advantage of it and get some stuff on-screen from it? In 2D, itwas easy... you just
used the same X axis as in your coordinates, and theY axis was the same only flipped upside down (since as you know, in codingit's easier if the Y axis goes down,
for calculation reasons). But howare we going to take this Z thing into account and make things look right?

The way we do it is by "Perspective Projection". Perspectiveis a pretty basic idea which you see continuously in your life.... Thingsfurther away from you look
smaller than things closer up. Pretty obviousfact that you've lived with throughout your existence. :)

Well all we have to do is turn that simple principle into math...

English way: "Things further away from you look smaller than things closer up."

Math way: "The projected size of any object in the eye is inversely proportional to its distance from the eye".

It's more "math-like", but still makes sense, right? Theyboth mean the same thing more or less, but the second phrase is more easilyturned into equation form... the
kind we can put into our code.

Okay, by our ASCII graph of the coordinate system, your eye lies rightalong our Z axis, staring down in the negative direction (the Z axis goesout of your screen,
and you're looking in). So an object's, or point's,distance from the eye is going to have a lot to do with the Z coordinateof that point.

So the first thing we need to settle is, if our eye (or in our case,the video screen) is sitting somewhere along the Z axis.... where alongthe Z axis is it? In truth, this is
entirely up to you... the screen isjust a "camera", and we're trying to find a convenient placefor it to sit. All we know is that it goes along the positive Z axis.

It turns out for calculation reasons that a very effective place toput the camera is at Z=256. You'll see why in a minute... (and no, noneof this is carved in stone.
Cameras can go anywhere and view in any direction,it's just that the math gets more complicated).

Okay, now we've got the camera at (0,0,256) and pointing in the negativedirection. That means that our Cartesian origin is exactly 256 units awayfrom us, right
along our line of sight. The next thing we need to makeup is a frame of reference for X and Y.... How big, visually, is a unitalong X or Y? Well we can use any set of
values we see fit, but just tomake it easier on the brain, we want to use something we're already usedto... something intuitive...

How about 1 pixel? It's easy to compute, since it's the same as yourscreen plane. Okay, so we know _somewhere_ viewable along this axis, Xand Y are going to
mean 1 pixel units.

But where along the Z axis? If things get larger as they get closerand smaller as they get further away, where's a good distance to say wherethe unit/pixel plane sits?

Heck, why not just use the Z=0 plane? After all, we know the originis perfectly visible (256 units down the -Z axis) so the origin is easilyin reach, and probably will
make it easy to calculate our perspective viewing.

Okay, we've got all we need to set up some perspective equations. Nowlet's do some simple examples in our heads so we can find out where theseequations are
coming from...

In 320x200 resolution (you can use any resolution you want, really,but this is just for demonstration), the center point of your screen isat (160,100). Makes sense,
and you're probably very used to this by now.Well, we're viewing along the -Z axis, so that's where our axis is goingdown. Right along the line of that pixel. No up,
no down, no left, no right.That's it.

Now we know that at Z=0, Every X and Y unit mean one pixel. So wherewould (5,5,0) be positioned? Well, it's the same as our screen plane, sothere's no
distance adjustments to do. It's 5 to the right of the origin,so ScrX=160+5 = 165. And it's 5 above the origin, which is in the oppositedirection of our "screen plane"
Y, so that's ScrY=100-5 = 95.You've got your projected point.

So what to do with points not on Z=0? Howsabout (5,5,128)? Well if wethink about it, Z=0 is 256 units distance from the screen. Well Z=128 is256-128 = 128
units distance from the screen. Exactly half of the distance.

Half the distance? That makes it twice the size! :-) So instead of adding5 to X and subtracting 5 from Y, you'd use 10 instead for each axis. Thatwould give us
screen coords (170,90) for this point. Same 3D X and Y asin the first example, but the fact that it's so much closer makes it lookfurther from the center of the
screen... the essence of perspective. :)

Now in general, our equations go something like this

ScrX = (Lens*X / Distance) + CenterX ScrY = CenterY - (Lens*Y /Distance)

I'll explain each piece. First, ScrY _would_ be the same basic equationas ScrX, except that we have to do the subtraction instead because Y goesdown onscreen,
not up. Anyway, we have a few variables here...

ScrX and ScrY are the screen point of the 3D point we're trying to project.I figure you guessed that one already. :-)

Distance is the distance of the point from the eye. As we discussedbefore, this has a _direct_ correlation to the Z coordinate of our 3D point.Now since we're on
the positive Z axis but viewing in the negative direction,and we know that at Z=0 the distance is 256, then any point where Z >0 will have a distance < 256.
Likewise, if Z < 0, the point willbe further away than the origin, and distance > 256. This makes distancea very simple value:

Distance = (256-Z)

Pretty easy, eh? :-) Now, Lens is a multiplier used to give your projectionthe right "field of view" that you want. You can manipulate Lensto give the projection a

very narrow range of vision, or to give it theclassic "wall-eye" view as well. But in general, a really messed-upfield of view will make people want to vomit. It's not
natural.

So we want to set it to something that will "feel" natural.Well, from our calculations and the fact that we want a unit/pixel relationshipat Z=0, we can get our Lens
factor immediately... 256. It's also a nicemultiplier, as you can use bit-shifts to get it, instead of a costly MULinstruction.

And finally, CenterX and CenterY are positive integers that match tothe center X and Y point onscreen; 160 and 100 in this case (the Centervalue for any resolution
is just the resolution/2, which makes sense).

So, from there, our equations boil down to

ScrX = (256*X / (256-Z)) + 160 ScrY = 100 - (256*Y / (256-Z)

The first thing you should recognize is that we don't need to calculate(256-Z) twice. Also, if you use fixed point, you can calculate the divideonly once as well, but
that's for a later article (I'm not focusing onoptimizations in this one). Anyway, let's work out one of our previousexamples using these equations and see if we get
the same result. How aboutthe (5,5,128) one... let's see...

Distance = 256-Z = 256-128 = 128

ScrX = (256*X / 128) + 160 = (256*5 / 128) + 160 = 10 + 160 = 170

ScrY = 100 - (256*Y / 128) = 100 - (256*5 / 128) = 100 - 10 = 90

Yup, same answer! And these equations go quite easily into code. :-)

Now you'll find that using straight integer math with these equationswill work well at first, but there are added advantages to using fixedpoint math as well. If you're
familiar with fixed point, you should havefew problems trying to adapt this to it, speeding things up in the process.If you're not familiar with it, don't sweat it; I'll
cover it in an upcomingarticle.

Section Three 2D Rotation

Before we can get more sophisticated 3D rotations going, we need totry it in two dimensions first... because 3D rotations are just based onthree 2D rotations, but
combined.

So how do we rotate something in 2D? How do we take any 2D point, giveit an angle to rotate by about the origin, and get it correctly to itsnew position? Well this
is where that Trig knowledge from the first articlecomes into play.

Everything about rotation involves Trig. Sine and Cosine are very muchyour friends here. And it's not that complicated, really... you can rotatein one plane with only
4 multiplies (other optimizations come later aswell).

So how do we go about this? Well, let's take it piece by piece. First,I'll assume the XY plane (the real one, where Y goes up) for this, as wetry to take a point and
rotate it.

A lot of docs, when trying to explain rotation, will give you the simpleequations for it but give you no clue as to how those equations came about.Several people

have asked me, "Hey, if and when you ever do a 3D tutorial,tell me how the heck you get those rotation equations, cuz I have no ideawhere those came from and
why they work."

Well, I can't quite tell you where they came from at first (like whothought of them), but I can replicate the ideas here and show you whatmakes sense to me. If it
makes sense to you to, then I guess it worked.:-)

Here's the idea...

Get out a piece of paper. No, don't worry, this isn't a quiz. ;)

On the paper, draw a pair of conventional XY coordinate axes, and thenlightly sketch a large circle on it. Make sure the circle is light; youdon't really need it for
much except placing a couple points.

After you draw the circle, put a point at about, say, 30 degrees (assuming0 degrees is to the right and the angles go counterclockwise). Then putanother point at
about 70 degrees, in the same fashion. We're going topretend that the first point is our original point, and that we're tryingto rotate it to the second point, our
destination... a rotation of 40 degreesabout the origin. The actual accuracy of the points doesn't matter; ifyou're a bit off, it's fine.

Now with each point, draw a triangle for that point. Each triangle'sthree sides are the X axis, the the line from the origin to the point,and the line from the point
straight down to the X axis. What you shouldhave now are two right triangles in the upper right quadrant of your XYplane, one being pretty upright (the destination
point's), and the othera bit more wide than tall.

Time for some labels... okay, for each triangle, label the line goingfrom the origin to the point as "R" (for radius). Since it'sthe same length for both triangles, we use
the same label. Now, on thefirst triangle (the short, wide one), label the side along the X axis "X",for that length. Likewise, label the line from the X axis up to the
pointas "Y" for that height.

For the second triangle (the tall one, for the 70 degree point), labelthe X length and Y height as "U" and "V", respectively,in a similar fashion.

Finally, we need two angles. In the angle between the X axis and thefirst, lower R side (30 degrees), label it í (called Phi). Then label theangle between the lower R
and the higher R (the one at 70 degrees) as é(called Theta).

There we go... we've got our drawing. :-) If my little walkthrough indrawing this has confused you to no end, either try it again from the beginning,or look at the
PCX in this supplement, with an image of the same diagramI'm describing.

Okay, so we have this drawing. Basically, what we know in the beginningis that we have this initial point at an unknown angle (we know it's 30degrees in this
example, but normally, you won't know that for arbitrarypoints), yet we know it has Cartesian coordinates (X,Y). What we want todo is pump X and Y through an
equation or two, along with the angle wewant to rotate by (which we labeled as Theta, and in this example is 40degrees), and find out its new coordinates, called
(U,V). So what equationsdo we use? Let's find out...

There are several convenient identities in Trigonometry that you canfind in pretty much every math textbook with Trig in it.... one of thoseidentities is called the "Law
of Sines", which goes like this...

Sin(à) Sin(á) Sin(â) ------ = ------ = ------ A B C

Where A, B, and C are the lengths of the sides of a triangle, and à,á, and â are the angles _directly opposite_ those sides...

/| /á| C / | / |A / | / | /à â| -------- B

It doesn't have to be a right triangle; it works for every trianglethere is. Granted, for our purposes, we _will_ be using our right triangles,and this will help us out.

Now if we use our first right triangle, the short one, and pretend thatR is our "C" of the triangle, by the fact that this is a righttriangle, we know that â is 90 degrees.
And the Sine of 90 is 1, whichgives us one very nice piece of math meat.

We only need to use one other side of our Law of Sines formula in thisexample, in this case, the A-à side. In our case, "A" is thesame as Y, and à is the same as í.
So we have a little mini-formula,

Sin(90) Sin(í) 1 Sin(í) ------- = ------ which means --- = ------ R Y R Y

Then, if you multiply each side by Y, it moves the Y to the left side,so

Y --- = Sin(í) R

This should all make sense so far, I hope. If you're looking at thediagram as you read this, it should clear things up a bit.

Okay, so we can see the relation between the angle í, and the sidesY and R. Well since í is across from Y, shouldn't we be able to have thesame kind of relation for
the other triangle, with V and R? The angle acrossfrom V is just í and é added together, so shouldn't that work?

Sure does. :-)

V --- = Sin(í+é) R

Okay, time for another nifty Trig identity (BTW, if you don't have amath book with all these identities in it, let me know... if enough peopleask for a listing, I'll type
up a quick reference list with identity equationsthat you can use. Just email to the address at the end, if you think you'dlike that :)

Anyway, another nice identity is that for any two angles à and á,

Sin(à+á) = Sin(à)*Cos(á) + Cos(à)*Sin(á)

So we sub that into our previous thing, and we have

V --- = Sin(í)*Cos(é) + Cos(í)*Sin(é) R

Multiply by R now, to get V (the destination point's X value that we'vebeen trying to find), and it's

V = R*Sin(í)*Cos(é) + R*Cos(í)*Sin(é)

Welp, last identity.... this one, taken from Polar coordinates. If you'vehad algebra, you've used Polar coordinates before. Well if you rememberthe way to convert a
polar point to Cartesian (I doubt you do, so I'llremind you... it's gonna take a while before you end up memorizing allthese darn formulas, trust me :) those
conversions are

X = R*Cos(Theta) *** Don't confuse these with our R, X, or Y! Y= R*Sin(Theta) They're just conversion equations ***

Well look at our V equation above... notice anything? We know Phi isan angle in the triangle that deals only with X and Y, which we know (sincethey're just your
first point and all). So can we drop those R*Sin(í) andR*Cos(í) parts and just sub in X and Y like you would do with Polar? Youbetcha.....

V = Y*Cos(é) + X*Sin(é) *** FINAL V EQUATION!!! :) ***

That's all we need! Hooray! :) We know X and Y, since we started withthose. And we know é, since it's the number of degrees we want to rotateby (in our
example, 40 degrees). So if we use this equation, we get theV value, which is the Y coordinate of the FINAL point. :)

Now we still need to get U (the final point's X coordinate). Luckily,the series of equations is the same almost, except one identity is different.I won't work out the
whole thing again, you can do that if you want. Buthere are the differences that you'll see. One, since we're doing the horizontalelement instead of vertical,

U --- = Cos(í+é) R

Now's Cosine's Sum of Angles formula is a little bit different thanSine's,

Cos(à+á) = Cos(à)*Cos(á) - Sin(à)*Sin(á)

which will end up giving us that subtraction instead of addition inthe end. If you keep working the equations the same as we did before, butwith this new identity, you
get the U equation too! :)

U = X*Cos(é) - Y*Sin(é) *** FINAL U EQUATION!!! ***

Summing up those equations into nice, happy, 2D rotation form.....

NewX = (OldX*Cos(Theta)) - (OldY*Sin(Theta)) NewY = (OldY*Cos(Theta))+ (OldX*Sin(Theta))

And there we have it! Note that I made it very clear as to the differencebetween the "Old" and "New" values. It's importantthat you do this, too. You don't want to
just use a value "X",for example.... because if you calculate the "new" X and endup using that instead of the "old" X in the second equation (forNewY), you don't get
the right rotation.

IN ROTATION, USE ONLY THE OLD VALUES UNTIL ALL THE NEW ONES ARE FOUND!

Once you have the final new X and Y values, _THEN_ replace the old pairwith the new pair, and go on your way. Make sure to keep the values separateuntil that
time.

BTW... As you look back at how I derived these rotation formulas, don'tfeel bad if you feel like you couldn't have derived them yourself... especiallyif you're just
beginning. I know I ran on these formulas blindly for overa year before I ended up losing them and was forced to recreate them againin this fashion. I couldn't have
done it earlier. It takes time, so ifyou feel like you're still in the dark... don't. Eventually you'll getthe hang of it all. :-)

Any more to 2D rotation? Nope, that's the whole of it. Before you tryout 3D rotation (explained in the next section), test out the above principlesin some of your
own code, by plotting a few pixels here and there and thenrotating them about the origin. It's not hard at all to turn the aboveformulas (formulae?) into code. Also, if
you need some help or are justplain curious, I've got some example source (in both Pascal and C, justlike last time) in this supplement, demonstrating this stuff. Feel
freeto check it out. :)

Okay, well, enough of this planar stuff.... on to 3D rotations! (Andrelax, there's not much more; you've done the bulk of the work already....)

_____Section Two - 3D Rotation

So what do we need to turn our rotations into 3D rotations? Not much,actually. There are many ways to do rotations in 3D, some simpler thanothers. The simplest
(and most common from what I've seen) way is to doit by using three 2D rotations, one for each axis.

The 2D rotations we did in the last section are on the XY plane. Butas you think about the XY plane in terms of 3D, the rotation takes on anothermeaning... it was
also a rotation ABOUT the Z axis. Meaning that we havethe Z axis, and whatever Z values the points may have, they stay the same,as we are rotating around that
axis itself. The only values that changein a rotation about any axis are the values of the two OTHER coordinates.

So a rotation about Z will affect X and Y, a rotation about X will affectY and Z, and a rotation about Y will affect Z and X. It's just one bigcycle...

So if we want to do a full all-axis 3D rotation, we just arrange threeback-to-back 2D rotations, one for each axis, like this...

NewY = (OldY*Cos(ThetaX)) - (OldZ*Sin(ThetaX)) ** X axis rotation ** NewZ = (OldZ*Cos(ThetaX)) + (OldY*Sin(ThetaX))

(Copy NewY and NewZ into OldY and OldZ)

NewZ = (OldZ*Cos(ThetaY)) - (OldX*Sin(ThetaY)) ** Y axis rotation ** NewX = (OldX*Cos(ThetaY)) + (OldZ*Sin(ThetaY))

(Copy NewZ and NewX into OldZ and OldX)

NewX = (OldX*Cos(ThetaZ)) - (OldY*Sin(ThetaZ)) ** Z axis rotation ** NewY = (OldY*Cos(ThetaZ)) + (OldX*Sin(ThetaZ))

(No copies needed, since we're done)

The reasons for mid-copies are like I said; for each axis rotation youneed to keep using the old values until both the new ones are done. Buteach axis's rotation is
independent of the other two... so after each pair,you need to update all the values before going on to the next axis. Youdon't want to use one axis's old values when
going into rotating aboutanother axis; that would be bad.

Once you've done all three axes, you should have your new point, completelyrotated about each angle as you wish (ThetaX, ThetaY, and ThetaZ).

One important point... the order in which you do these axes DOES makea difference. Rotating in an X-Y-Z sequence will not give you the sameresults as rotating in
a Z-X-Y sequence, etc. Now, for your engine at thispoint, all you're probably concerned about is looks, i.e. that your objectis rotating and you can see it rotating.
Since that's the case, it reallydoesn't matter for the moment which order you do things in. It's the appearancethat counts. But later on, when you get into more
complex issues that involvemore things than just a set of points, you'll want to keep your rotationorder consistant. I just use X-Y-Z because it's pretty natural. :-)

I'm not going to get into optimizations of this rotation material untilanother time, but I can give you a hint or two now... first, you'll noticethat right now it's at 12
multiplies for a full rotation (4 for each axis).But it turns out you can reduce it to at least 9 multiplies, by precalculatinga few values at the beginning of each frame and

getting a final 3x3 matrixfor the actual point rotations themselves (if you don't know what I meanby matrix, don't worry about it at the moment; we'll get into
matriceslater on). It's something to look into, if you're curious and feel liketinkering with the math a bit.

Also, once again, this method of rotation is only one way to rotate.There are other ways, sometimes involving other coordinate systems, thatcan be more efficient on
occasion as well. You'll discover those in time(and probably in some of the later articles :) But for now, this I thinkis the simplest way to begin... get these concepts
down first, and drillthem into your brain. You'll know when to switch gears when the time comes.