CS 450/550 -- Fall Quarter 2020

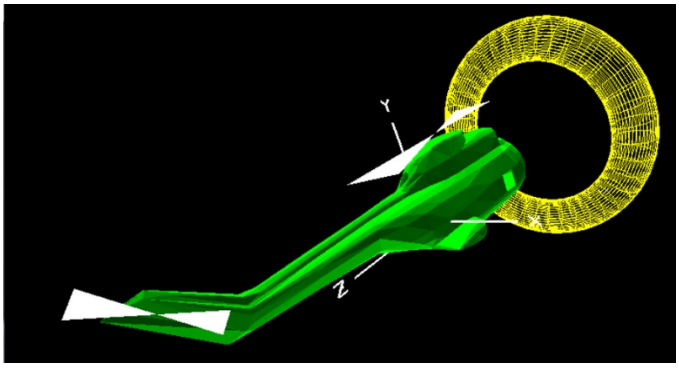
Project #2

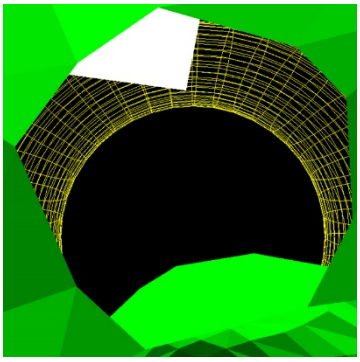
100 Points

Due: October 13

Animate a Helicopter!

This page was last updated: July 26, 2020





Introduction

This project is to animate a helicopter and look at it in two different 3D views.

Requirements:

1. Draw a helicopter. (Don't worry -- this won't be as hard as it sounds.)
2. The helicopter's 2 blades must be scaled properly.
   The helicopter's 2 blades must be oriented properly.
   The helicopter's 2 blades must be rotating properly.
3. Allow two views: an "Outside" view of the entire scene and an "Inside" view from the helicopter cockpit. Toggle between them with a pop-up menu. (You can also use keyboard hits, but in-addition-to, not instead-of.) For each view, use a different call to **gluLookAt( )** to position the eye. A good location for your Inside Eye is (-0.4, 1.8, -4.9) .
4. Keep the same rotate and scale features as we've used before, ***but only in the Outside View***.
5. Use **gluPerspective( )**, not **glOrtho( )**.

6. Put some sort of 3D scene for your Inside Eye to see, looking outside the helicopter. One or more GLUT wireframe objects might work well. Or, whatever you did in the first project. Or some sort of colored grid. Or, all of these things.
7. Use the graphics programming strategy where the **Display( )** function looks at a collection of global variables and draws the scene correctly. The other parts of the program simply set the global variables and post a redisplay.
8. The code for creating the helicopter geometry is shown below in the **Geometry** section. The helicopter body extends along the Z axis. The heicopter cockpit points in -Z.
9. There must be two blades drawn on the helicopter body: a large blade on the roof, oriented in the X-Z plane, and a small one on the tail, oriented in the Y-Z plane. Create a single blade display list and then transform and instance it twice to get the two blades in the scene. The code for creating the single blade geometry is shown below in the **Geometry** section. The single blade is in the X-Y plane, centered at **(0.,0.,0.)**, with a radius of **1.0**.
10. The top helicopter blade is to have a radius of **5.** and be attached at: **(0.,2.9,-2.)**. The rear helicopter blade is to have a radius of **1.5** and be attached at: **(.5,2.5,9.)**.
11. Both blades need to rotate. The rotation rate is up to you, but the *smaller one must rotate **3X** as fast as the larger one*. That is, for whatever amount you advance the rotation the large blade's rotation angle, advance the small blade's rotation angle **3X** that amount.
12. Pay close attention to the overall transformation sequence. You can re-use some of the transformations that you have already created by using **glPushMatrix( )** and **glPopMatrix( )**.
13. Parameterize your scene as much as you can with **#define**'s or **const**'s, It makes it easier to make changes later.
14. Use GLUT pop-up menus for all options.

Positioning Everything:

- Remember how we did it in class: write down in words what you want to happen and then program it backwards.
- For the helicopter, you might have a word-transformation sequence like this:
    1. If you are in the Outside view, apply the Xrot, Yrot, and Scale transformations as before.
    2. Assign a good eye, look, and up position with **gluLookAt**.

  Then draw the helicopter.
- For a blade, you might have a word-transformation sequence like this:
    1. Scale the blade to be the right size.
    2. Rotate the blade to be in its proper orientation.
    3. Rotate the blade by its current spinning rotation angle.
    4. Translate the blade to where it needs to be on the helicopter.
    5. If you are in the Outside view, apply the Xrot, Yrot, and Scale transformations as before.
    6. Assign a good eye, look, and up position with **gluLookAt**.

  Then draw the blade.
  and so on.

Supplying the Geometry

- Click here to get the file called **heli.550**. Move it into your project workspace directory and then **#include** it in your C/C++ code just after your list of global variables.
- The **heli.550** file has the structure definitions in it, so you won't need to define them yourself.
- If you want to draw a wireframe helicopter, use the following code when you create your helicopter display list:

```
int i;
struct edge *ep;
struct point *p0, *p1;

glPushMatrix( );
glTranslatef( 0., -1., 0. );
glRotatef(  97.,   0., 1., 0. );
glRotatef( -15.,   0., 0., 1. );
glBegin( GL_LINES );
	for( i=0, ep = Heliedges; i < Helinedges; i++, ep++ )
	{
		p0 = &Helipoints[ ep->p0 ];
		p1 = &Helipoints[ ep->p1 ];
		glVertex3f( p0->x, p0->y, p0->z );
		glVertex3f( p1->x, p1->y, p1->z );
	}
glEnd( );
glPopMatrix( );
```

- If you want to draw a polygon helicopter, use the following code when you create your helicopter display list:

```
int i;
struct point *p0, *p1, *p2;
struct tri *tp;
float p01[3], p02[3], n[3];

glPushMatrix( );
glTranslatef( 0., -1., 0. );
glRotatef(  97.,   0., 1., 0. );
glRotatef( -15.,   0., 0., 1. );
glBegin( GL_TRIANGLES );
	for( i=0, tp = Helitris; i < Helintris; i++, tp++ )
	{
		p0 = &Helipoints[ tp->p0 ];
		p1 = &Helipoints[ tp->p1 ];
		p2 = &Helipoints[ tp->p2 ];

		// fake "lighting" from above:

		p01[0] = p1->x - p0->x;
		p01[1] = p1->y - p0->y;
		p01[2] = p1->z - p0->z;
		p02[0] = p2->x - p0->x;
		p02[1] = p2->y - p0->y;
		p02[2] = p2->z - p0->z;
		Cross( p01, p02, n );
		Unit( n, n );
		n[1] = fabs( n[1] );
		n[1] += .25;
		if( n[1] > 1. )
			n[1] = 1.;
		glColor3f( 0., n[1], 0. );

		glVertex3f( p0->x, p0->y, p0->z );
		glVertex3f( p1->x, p1->y, p1->z );
		glVertex3f( p2->x, p2->y, p2->z );
	}
glEnd( );
glPopMatrix( );
```

- In the **glColor3f( )** call, take whatever R, G, B triple you want the helicopter to have and scale them all by *n[1]*. In the above example, Joe Graphics wanted a green helicopter.
- To create the single blade, from which the other two blades will be instanced, use the following code when you create your blade display list:

```
// blade parameters:

#define BLADE_RADIUS		1.0
#define BLADE_WIDTH		0.4

// draw the helicopter blade with radius BLADE_RADIUS and
//	width BLADE_WIDTH centered at (0.,0.,0.) in the XY plane

glBegin( GL_TRIANGLES );
	glVertex2f(  BLADE_RADIUS,  BLADE_WIDTH/2. );
	glVertex2f(  0., 0. );
	glVertex2f(  BLADE_RADIUS, -BLADE_WIDTH/2. );

	glVertex2f( -BLADE_RADIUS, -BLADE_WIDTH/2. );
	glVertex2f(  0., 0. );
	glVertex2f( -BLADE_RADIUS,  BLADE_WIDTH/2. );
glEnd( );
```

Getting Started:

Not sure where to start? Read on!

1. Draw the helicopter at the origin.
2. Draw at least one other scene element somewhere out along -Z. This is where the Inside View will be looking.
3. Start by using the stationary Outside View to view the scene. Give gluLookAt( ) some good values.
   Play with these so that when your program starts up, your are seeing the helicopter and your whole scene from a good angle.
4. Pay careful attention to:
   - Make the near clipping plane location very small, e.g., 0.1
   - Make the far clipping plane location big-ish (not huge), e.g., 1000.
   - Be sure to set some non-BLACK color right before drawing things.
   - Be careful how far the eye is from the helicopter. Too close and you might get the helicopter clipped off. Too far and the helicopter will look like a little fly speck.
5. After that works, designate **Animate( )** as the Idle Function in InitGraphics( ). Have a global variable, say **BladeAngle**, that gets incremented in **Animate( )**. Use that variable in **Display( )** to rotate the blades. Be sure that **Animate( )** posts a re-display.
6. A way that I really like to do the animation is to put this code in Animate( ):

```
float Time;
#define MS_IN_THE_ANIMATION_CYCLE		10000
. . .
int ms = glutGet( GLUT_ELAPSED_TIME );		// milliseconds
ms  %=  MS_IN_THE_ANIMATION_CYCLE;
Time = (float)ms  /  (float)MS_IN_THE_ANIMATION_CYCLE;	// [ 0., 1. )
```

where **Time** is a global floating-point variable and **MS_IN_THE_ANIMATION_CYCLE** is how many milliseconds are in the animation cycle. This sets **Time** to be between 0. and 1., which you can then use to set animation parameters. The advantage of this is that you will get the same number of milliseconds in the animation cycle regardless of how fast or slow a system you run this on.

7. After that works, add the Inside View by testing what view mode you are in and then using a different call to **gluLookAt( )**. Don't use Xrot, Yrot, and Scale if you are in the Inside Mode.

Those Vector-Manipulation Functions

The lighting of the helicopter surfaces use two functions, **Cross( )** and **Unit( )**. They are included already in your sample code.

A Debugging Suggestion:

- One thing that has always helped Joe Graphics is to have a "freeze" option, toggled with the '**f**' key. This freezes the animation so you can really look at your helicopter and see if it is being drawn correctly. Remember what the current freeze status is with a boolean global variable:

```
bool	Frozen;
```

- Set **Frozen** to *false* in **Reset( )**. Then, freezing the animation is just a matter is setting the Idle Function to **NULL**. To un-freeze it, set the Idle Function back to **Animate( )**. So, in the **Keyboard( )** callback, you could say something like:

```
case 'f':
case 'F':
	Frozen = ! Frozen;
	if( Frozen )
		glutIdleFunc( NULL );
	else
		glutIdleFunc( Animate );
	break;
```

Turn-in:

Use the Teach system to turn in:

1. Your .cpp file
2. A one-page PDF with a title, your name, your email address, a nice screen shot from your program, and the link to the Kaltura video demonstrating that your project does what the requirements ask for. Narrate your video so that you can tell us what it is doing.

Bonus Days:

Each of you has been granted five total Bonus Days, which are no-questions-asked one-day project extensions, but no more than **2** Bonus Days may be applied to any one project. Hint: Bonus Days will likely be worth a lot more to you late in the quarter than they are worth to you early in the quarter!

Grading:

Feature	Points
Correctly draw the helicopter body	15
Correctly scale the blades	20
Correctly position the blades	20
Correctly rotate the blades	20
Recognizable Inside View	25
Potential Total	**100**