Oregon State University

CS 450/550 -- Fall Quarter 2023

Project #2

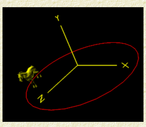
100 Points

Due: October 17

Using Transformations to Animate a Carousel Horse!

This page was last updated: September 12, 2023

Introduction



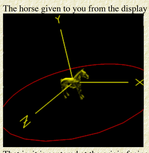
You don't have to be a kid to enjoy a ride on a carousel. (Also called a Merry-Go-Round.) But, you do have to understand computer graphics transformations to animate one. That's where you come in.

Learning Objective:

When you are done with this assignment, you will understand how to compound transformations to make the motion of some objects depend on the motion of other objects. This is a key for creating many of the typical animated scenes you see in games and movies.

Instructions:

1. Draw a horizontal circle with radius 2.0 in the X-Z plane (see above) to show the path of the horse.
2. Draw a carousel horse into a display list. Don't worry -- this won't be as hard as it sounds. The code for creating the horse geometry is shown below in the **Geometry** section. The horse's hooves are in the X-Z plane. The top of the horse points up in +Y. The horse's head faces in +X.
3. The horse given to you from the display list looks like this:

   That is, it is centered at the origin facing in the +X direction. It is up to you to make it animate properly from there.
4. Each horse will have 4 transformations, not necessarily in this order:
   - The horse needs to translate up and down
   - The horse needs to revolve in a circle
   - The horse needs to be translated from the origin to the outside of the circle
   - The horse needs to rock back and forth (like a rocking horse). If this was an airplane, we would call this motion "pitching".
5. Allow two views: an "Outside" view of the entire scene and an "Inside" view from the center of the carousel looking outward. Be able to switch between them in your video. (You could use a pop-up menu or a keyboard hit) For each view, use a different call to **gluLookAt( )** to position the eye.
6. Keep the same Xrot, Yrot, and Scale features as we've used before, ***but only in the Outside View***. Do not use Xrot, Yrot, and Scale in the Inside View.
7. Use **gluPerspective( )**, for both views. Only allow a switch to **glOrtho( )** in the Outside view.
8. 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 set the global variables and post a redisplay.
9. When you get one horse to behave correctly, there is a +10 point Extra Credit if you add 3 more horses, each 90° apart on the circle, each translating and rocking out of phase with the others.

Animating the Horse:

Remember how we did it in class: write down in words what transformations you want to have happen and then program it backwards. If you have a stuffed animal handy, you might imagine a set of axes on it and then figure out what motions it would need to have to act like a carousel horse.

Supplying the Geometry

- Right-click here to get the file called **CarouselHorse0.10.550**. Save it into your project workspace directory and then **#include** it in your C/C++ code just after your list of global variables.
- (The "0.10" signifies that I removed 90% of the detail from the original horse model to keep the size of the file down. I still have the full model if you would like to use that instead.)
- The **CarouselHorse0.10.550** file has the structure definitions in it, so you won't need to define them yourself.
- If you want to draw a wireframe horse, use the following code when you create your horse display list in **InitLists( )**:

```
	WireHorseList = glGenLists( 1 );
	glNewList( WireHorseList, GL_COMPILE );
		glPushMatrix( );
			glRotatef(90.f, 0., 1., 0.);
			glTranslatef( 0., -1.lf, 0.f);
			glColor3f( 1.f, 1.f, 0.f);	// yellow
			glBegin( GL_LINES );
				for( int i=0; i < HORSEnedges; i++ )
				{
					struct point p0 = HORSEpoints[ HORSEedges[i].p0 ];
					struct point p1 = HORSEpoints[ HORSEedges[i].p1 ];
					glVertex3f( p0.x, p0.y, p0.z );
					glVertex3f( p1.x, p1.y, p1.z );
				}
			glEnd( );
		glPopMatrix( );
	glEndList( );
```

- If you want to draw a polygon horse, use the following code when you create your horse display list in **InitLists( )**:

```
	HorseList = glGenLists( 1 );
	glNewList( HorseList, GL_COMPILE );
		glPushMatrix( );
			glRotatef(90.f, 0., 1., 0.);
			glTranslatef( 0., -1.lf, 0.f);
			glBegin( GL_TRIANGLES );
				for( int i = 0; i < HORSEntris; i++ )
				{
					struct point p0 = HORSEpoints[ HORSEtris[i].p0 ];
					struct point p1 = HORSEpoints[ HORSEtris[i].p1 ];
					struct point p2 = HORSEpoints[ HORSEtris[i].p2 ];

					// fake "lighting" from above:

					float p01[3], p02[3], n[3];
					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] = (float)fabs( n[1] );
					// simulating a glColor3f( 1., 1., 0. ) = yellow:
					glColor3f( 1.f*n[1], 1.f*n[1], 0.f*n[1]);

					glVertex3f( p0.x, p0.y, p0.z );
					glVertex3f( p1.x, p1.y, p1.z );
					glVertex3f( p2.x, p2.y, p2.z );
				}
			glEnd( );
		glPopMatrix( );
	glEndList( );
```

  In the **glColor3f( )** call, take whatever R, G, B triple you want the horse to have and scale them all by *n[1]*. In the above example, this is how you get yellow.

Getting Started:

Not sure where to start? Read on!

1. Draw the circle the horse will be traveling along. The circle should be in the X-Z plane, centered at the origin, with a radius of 2.0
2. Call up the horse display list with no transformations. This will put the horse at the origin.
3. Start by using the stationary Outside View to view the scene. Give gluLookAt( ) some good values. The ones from the sample code will be a good start.

   Play with these so that when your program starts up, your are seeing the horse and your whole scene from a good angle.
4. Provide these 4 transformations for the horse, not necessarily in this order:
   - The horse needs to translate up and down
   - The horse needs to revolve in a circle
   - The horse needs to be translated from the origin to the outside of the circle
   - The horse needs to rock back and forth (like a rocking horse).
5. Your sample code already has this code in Animate( ):

```
	int ms = glutGet(GLUT_ELAPSED_TIME);
	ms %= MS_PER_CYCLE;                     // makes the value of ms between 0 and MS_PER_CYCLE-1
	Time = (float)ms / (float)MS_PER_CYCLE;	// makes the value of Time between 0. and slightly less than 1.
```

   where **Time** is a global floating-point variable and **MS_PER_CYCLE** is how many milliseconds are in the animation cycle. The sample code set this to 10000 (10 seconds), but you can change it.

   This code 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.
6. After that works, add the Inside View which will look outward from the center of the circle. Your code should test what view mode you are in and, if you are in the Inside mode, use a different call to **gluLookAt( )**. Don't use Xrot, Yrot, and Scale if you are in the Inside Mode.

Those Vector-Manipulation Functions

The code to light the horse's surfaces uses two functions, **Cross( )** and **Unit( )**. They are in your sample code already.

A Debugging Suggestion:

One thing that has always helped Joe Graphics debug animation programs is to have a "freeze" option, toggled with the **'f'** key. This freezes the animation so you can really look at your horse and propellers and see if they are being drawn correctly. Remember what the current freeze status is with a boolean global variable. Set it 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, the whole thing could look like this:

```
// a global:
bool Frozen;

// in Reset( ):
	Frozen = false;

// in Keyboard( ):
	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 short PDF report containing:
   - Project number and title
   - Your name
   - Your email address
   - A description of what you did to get the display you got
   - A couple of cool-looking screen shots from your program
   - The link to the Kaltura video demonstrating that your project does what the requirements ask for. If you can, we'd appreciate it if you'd narrate your video so that you can tell us what it is doing.
3. **Be sure that your video's permissions are set to *unlisted*.** The best place to set this is on the OSU Media Server.
4. A good way to test your video's permissions is to ask a friend to try to open the same video link that you are giving us.
5. The video doesn't have to be made with Kaltura. Any similar tool will do.

Grading:

Feature	Points
Correctly draw a horse body with the "fake lighting"	20
The horse moves in a circle correctly	20
The horse bobs up and down correctly	20
The horse rocks (pitches) correctly	20
Able to switch views	20
Extra Credit: Able to animate 4 horses correctly	10
Potential Total	**110**

Acknowledgement:

Thanks to free3d.com for the free horse model!