Introduction:

The goal of this project is to create an animated scene of Bézier curves.
1. It must be in 3D (i.e., Z cannot be constant).
2. It can be cubics (like we looked at in class), but it can also be 2nd order, 4th order, 5th, etc. That is, feel free to experiment with a different number of control points per curve.
3. You must have at least 5 curves.
4. The curves must somehow be "different" from each other.
5. That could mean rotated, scaled, different points, etc. Just translated is not enough.

Requirements:
1. Create a scene of at least five 3D Bézier curves.
2. What the scene looks like is up to you. 10 of the 100 project points are given if you deliberately make it look like something at least somewhat recognizable. You lose those 10 points if it looks like you created this scene by accident.
3. How you color the scene is up to you.
4. How you animate the scene is up to you.
5. How many points you use to draw each curve is up to you. (I.e., the Δt.) But, use enough to make the curves look smooth.

6. Animate each curve by changing the position of one (or more) points per curve based on the Time variable you have been creating in the last couple of projects.

7. Be able to turn the control points on and off.

8. Be able to turn the control lines on and off. (A "control line" is a line strip connecting the curve's first endpoint point to the control points in order, to the last endpoint.)

9. Enable the following keys:
   
   "f" Freeze/un-freeze the animation

10. (This isn't a requirement, but it always helps to debug animations.)

**Turn-in:**
Use the Teach system to turn in your:

1. Your PDF report, describing what you did and where we can find your video.
2. Be sure your video makes it obvious that you are doing this in 3D!
3. .cpp file

**Possible Way to Organize the Data**

```c
struct Point
{
    float x0, y0, z0; // initial coordinates
    float x, y, z;   // animated coordinates
};

struct Curve
{
    float r, g, b;
    Point p0, p1, p2, p3;
};

Curve Curves[NUMCURVES];    // if you are creating a pattern of curves
Curve Stem;                  // if you are not
```

**Reminder: The Cubic Bézier Equation is:**

\[
P = (1.-t)^3P_0 + 3t(1.-t)^2P_1 + 3t^2(1.-t)P_2 + t^3P_3
\]

\(0. \leq t \leq 1.\)

**Rotating a Point an Angle about the X Axis Around a Center**

```c
void RotateX(Point *p, float deg, float xc, float yc, float zc)
{
    float rad = deg * (M_PI/180.f); // radians
```
float x = p->x0 - xc;
float y = p->y0 - yc;
float z = p->z0 - zc;
float xp = x;
float yp = y*cos(rad) - z*sin(rad);
float zp = y*sin(rad) + z*cos(rad);
p->x = xp + xc;
p->y = yp + yc;
p->z = zp + zc;
}

Rotating a Point an Angle about the Y Axis Around a Center

void 
RotateY( Point *p, float deg, float xc, float yc, float zc )
{
    float rad = deg * (M_PI/180.f);         // radians
    float x = p->x0 - xc;
    float y = p->y0 - yc;
    float z = p->z0 - zc;

    float xp =  x*cos(rad) + z*sin(rad);
    float yp =  y;
    float zp = -x*sin(rad) + z*cos(rad);

    p->x = xp + xc;
p->y = yp + yc;
p->z = zp + zc;
}

Rotating a Point an Angle about the Z Axis Around a Center

void 
RotateZ( Point *p, float deg, float xc, float yc, float zc )
{
    float rad = deg * (M_PI/180.f);         // radians
    float x = p->x0 - xc;
    float y = p->y0 - yc;
    float z = p->z0 - zc;

    float xp = x*cos(rad) - y*sin(rad);
    float yp = x*sin(rad) + y*cos(rad);
    float zp = z;

    p->x = xp + xc;
p->y = yp + yc;
p->z = zp + zc;
}

Drawing a Bézier Curve

gLineWidth( 3. );
glColor3f( r, g, b );
gBegin( GL_LINE_STRIP );
  for( int it = 0; it <= NUMPOINTS; it++ )
  {
    float t = (float)it / (float)NUMPOINTS;
    float omt = 1.f - t;
    float x = omt*omt*omt*p0.x + 3.f*t*omt*omt*p1.x + t*t*t*p3.x;
    float y = omt*omt*omt*p0.y + 3.f*t*omt*omt*p1.y + t*t*t*p3.y;
    float z = omt*omt*omt*p0.z + 3.f*t*omt*omt*p1.z + t*t*t*p3.z;
    glVertex3f( x, y, z );
  }
  glEnd( );
gLineWidth( 1. );

How Did Jane Graphics Get That Nice Color Progression?

She used the hue-saturation-value color scale and the HsvRgb( ) function that is included in your sample code. As long as you are using angles to locate points, you can use that same angle to assign a hue and then turn it into an RGB. See the HSV slide of the Getting Started notes.

Grading:

<table>
<thead>
<tr>
<th>Item</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>At least 5 stationary 3D curves</td>
<td>20</td>
</tr>
<tr>
<td>At least 5 animated 3D curves</td>
<td>30</td>
</tr>
<tr>
<td>Turn the control points on and off</td>
<td>20</td>
</tr>
<tr>
<td>Turn the control lines on and off</td>
<td>20</td>
</tr>
<tr>
<td>Make the scene look somewhat recognizable</td>
<td>10</td>
</tr>
<tr>
<td>Potential Total</td>
<td>100</td>
</tr>
</tbody>
</table>