Introduction:
The goal of this project is to create an animated scene of Catmull-Rom curves.

1. It must be in 3D (i.e., Z cannot be constant).
2. You must have at least 5 curves.
3. Each curve can have any number of points defining it (>=6)
4. Pick a reasonable number of vertices for each sub-curve so that it looks smooth as t goes from 0. to 1.
5. The curves must somehow be "different" from each other. That could mean rotated, scaled, different points, etc. But, just translated is not good enough.

Requirements:
1. Create a scene of at least five 3D Catmull-Rom 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.
3. How you color the scene is up to you.  (It is helpful to us if each curve is a different color.)
4. How you animate the scene is up to you.
5. How many vertices you use to draw each sub-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 using in the last couple of projects.
7. Be able to turn the drawing of the points on and off.
8. Enable the following keys:
   `f` Freeze/un-freeze the animation

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!
A Possible Way to Organize the Data

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

struct Curve
{
    float r, g, b;
    Point *points;
    int count;
};

Curve Curves[NUMCURVES];  // if you are creating a pattern of curves

Reminder: the Catmull-Rom Sub-curve Equation is:

\[ P(t) = 0.5 \cdot \left[ 2 \cdot P_1 + t \cdot (-P_0 + P_2) + t^2(2 \cdot P_0 - 5 \cdot P_1 + 4P_2 - P_3) + t^3(-P_0 + 3P_1 - 3P_2 + P_3) \right] \]

\[ 0 \leq t \leq 1. \]

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

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);
}

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

```c
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;
}
```

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>Draw at least 5 3D curves</td>
<td>40</td>
</tr>
<tr>
<td>Animate those 3D curves</td>
<td>30</td>
</tr>
<tr>
<td>Turn the curves on and off</td>
<td>10</td>
</tr>
<tr>
<td>Turn the points on and off</td>
<td>10</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>

**Be sure it is obvious from your video that the curves pass through their points (except the first and the last)!**