CIS 441/541: Project #1F
Due by October 30th, 2016 (meaning 6am October 31st)
Worth 8% of your grade

Instructions

1) Download the following three files and incorporate them into your program. You are welcome to keep them as separate files if you are comfortable with CMake, but I am expecting that most of you will cut-n-paste their contents into your 1E code
    a. Download camera.cxx. It has a definition of the Camera class and also a method for generating Camera positions ("GetCameraPosition").
    b. Download matrix.cxx, which has my matrix class.
    c. Download reader1F.cxx. It no longer projects points to device space for you.
    d. (note: the shading parameters are the same as 1E)
2) Continue using the geometry file "proj1e_geometry.vtk".
3) Note that the output image is 1000x1000. You should initialize the buffer to be black (0,0,0). This was done for you in previous projects, so just make sure that code didn't go anywhere. Keep in mind you will be doing multiple renderings and need to initialize the color and z-buffers for each rendering.
4) Update your CalculateShading function to use the correct view direction. You will do this by calculating the view direction in world space (vertex position minus camera position), calculating the shading factor, and then placing the shading factor on the vertex. You can then LERP the shading variable over the triangle and apply the final shading right before you deposit the fragment color into the buffer.
5) Generate the correct camera positions for:

    Camera c1 = GetCamera(0, 1000);
    Camera c2 = GetCamera(250, 1000);
    Camera c3 = GetCamera(500, 1000);
    Camera c4 = GetCamera(750, 1000);
    (I recommend you do all 1000 positions and make a movie out of the result, but that's up to you.)

IMPORTANT: 1E had Z=-1 as the front. 1F has Z=+1 as the front. This means your depth test needs to switch from "<" to ">" when considering a pixel. I apologize for this change.

Note that differencer will no longer produce perfect outputs. If you get every pixel different, then your program is wrong. But if you ~20 pixels (or less) different, then you should declare victory. If you have hundreds or thousands of pixels difference, then you probably have a problem with your shading.

When you are done, submit the following to Blackboard:

- your code
- your four frames (from Camera c1, c2, c3, and c4)

- a screen shot of differencer congratulating you for each of the 4 frames

In terms of grading, expect less than half credit if you turn in an incorrect solution … I prefer a correct solution late (half credit) to an incorrect solution on time.

Tips:

(1) All vertex multiplications use 4D points. Make sure you send in 4D points for input and output, or you will get weird memory errors.
   a. Also don't forget to divide by "w"

(2) Your Phong lighting in 1E was in screen space and assumed a view of (0,0,-1). You will now calculate the shading (per vertex) in world space, i.e., before applying any camera transform. You will also interpolate the shading value per fragment as you rasterize. (This repeats point 4 from the instructions, and is just saying the same thing in a different way.)

(3) People often get a matrix confused with its transpose. Use the method Matrix::Print() to make sure the matrix you are setting up is what you think it should be. Also, remember the points are left multiplied, not right multiplied.
   a. This is opposite the notes in some cases, where we were often using right multiplication.

(4) Regarding multiple renderings:
   a. Don't forget to initialize the screen between each render
   b. If you modify the triangle in place to render, don't forget to switch it back at the end of the render

Here are the outputs for an example camera and points:

```
near = 5;
far = 200;
angle = M_PI/6;
position[0] = 0;
position[1] = 40;
position[2] = 40;
focus[0] = 0;
focus[1] = 0;
focus[2] = 0;
up[0] = 0;
up[1] = 1;
up[2] = 0;
```

```
Camera Frame: U = 1, 0, 0
Camera Frame: V = 0, 0.707107, -0.707107
Camera Frame: W = 0, 0.707107, 0.707107
Camera Frame: O = 0, 40, 40
Camera Transform
(1.0000000 0.0000000 0.0000000 0.0000000)
(0.0000000 0.7071068 0.7071068 0.0000000)
(0.0000000 -0.7071068 0.7071068 0.0000000)
(0.0000000 0.0000000 -56.5685425 1.0000000)
View Transform
(3.7320508 0.0000000 0.0000000 0.0000000)
(0.0000000 3.7320508 0.0000000 0.0000000)
(0.0000000 0.0000000 1.0512821 -1.0000000)
(0.0000000 0.0000000 10.2564103 0.0000000)
Total Transform
(1866.0254038 0.0000000 0.0000000 0.0000000)
(-353.5533906 965.9258263 0.7433687 -0.7071068)
(-353.5533906 -1673.0326075 0.7433687 -0.7071068)
(28284.2712475 28284.2712475 -49.2130831 56.5685425)
Transformed 0, 36.4645,36.4645, 1 to 500, 500,1
Transformed 0, -101.421,-101.421, 1 to 500, 500,-1
Transformed V0 from (1.11111,7.57576,-9.07897) to (535.976, 881.312, -0.873317)
Transformed V1 from (0.968446,7.57576,-8.9899) to (531.391, 879.688, -0.873122)
Transformed V2 from (1.11111,7.46665,-8.9899) to (535.967, 876.682, -0.87336)
```

At a high level, your code will be something like:

```
vector<Triangle> t = GetTriangles();
AllocateScreen();
for (int i = 0 ; i < 1000 ; i++)
{
   InitializeScreen();
   Camera c = GetCamera(i, 1000);
   TransformTrianglesToDeviceSpace(); // involves setting up and applying
matrices … if you modify vector<Triangle> t, remember to undo it later
   RenderTriangles()
   SaveImage();
}
```