Your Surrounding World

Benjamin Carter and Josh Canode

Grand Canyon University

CST-305: Computer Graphics
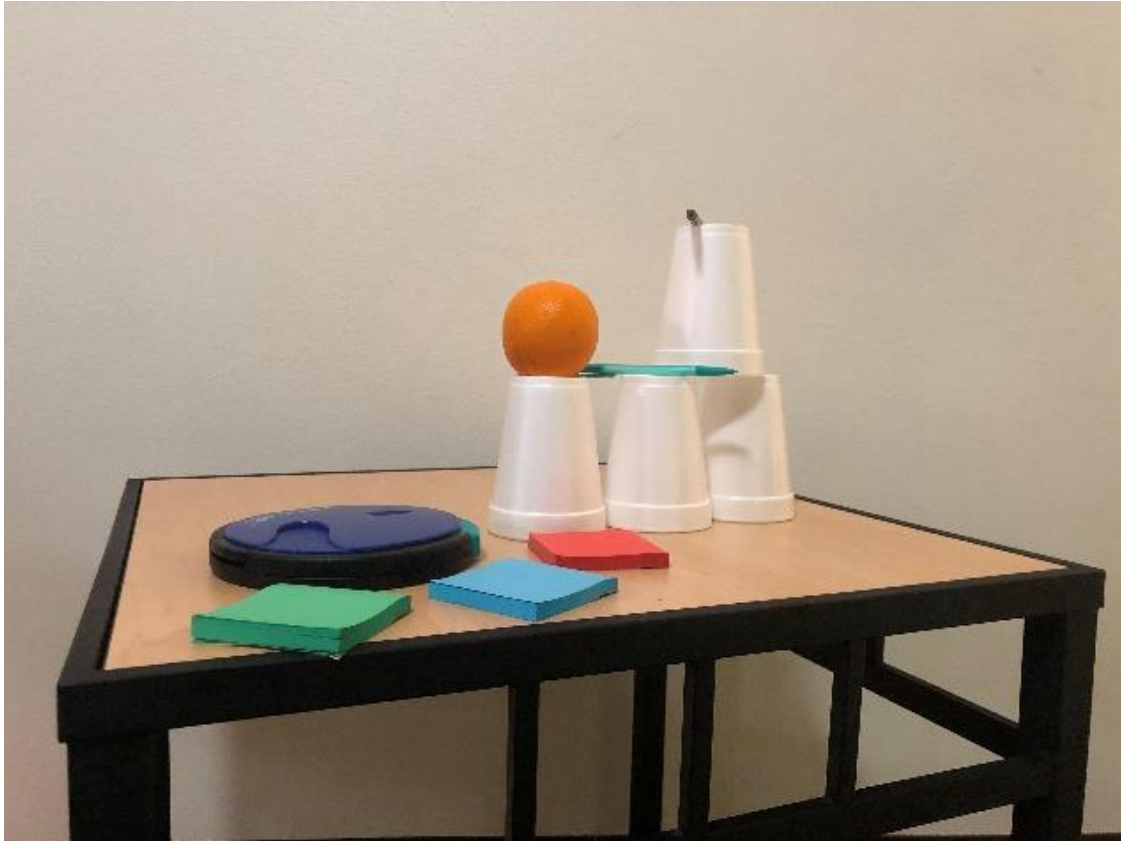
Professor Ricardo Citro

October 14th, 2023

Project Description

Our image involves a selection of objects on a table. There are several Styrofoam cups, an orange, packages of sticky notes, and a game. The game is a device used in the word game "Catch Phrase". This document outlines the various objects that will be recreated in OpenGL.


Main Object List

The following have been identified as the main objects in the scene. The reason for choosing these is obvious, these are all the objects in the scene, and are relatively equally prominent in the scene.

1. Cup 1
2. Cup 2
3. Cup 3
4. Cup 4
5. Green sticky notes
6. Blue sticky notes
7. Red sticky notes
8. Catch Phrase game

9. Table
10. Green Pen
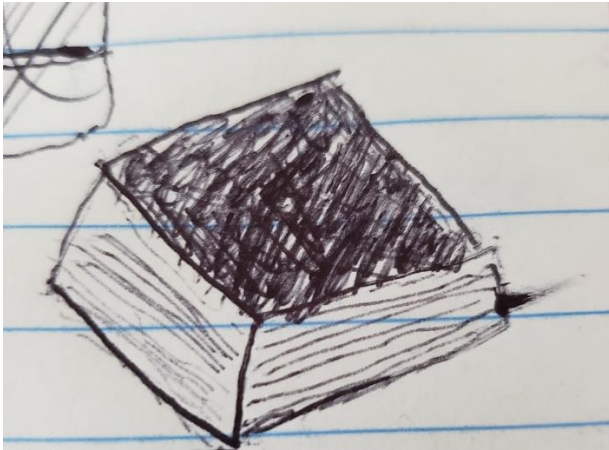11. Black Pen
12. Orange
13. Wall
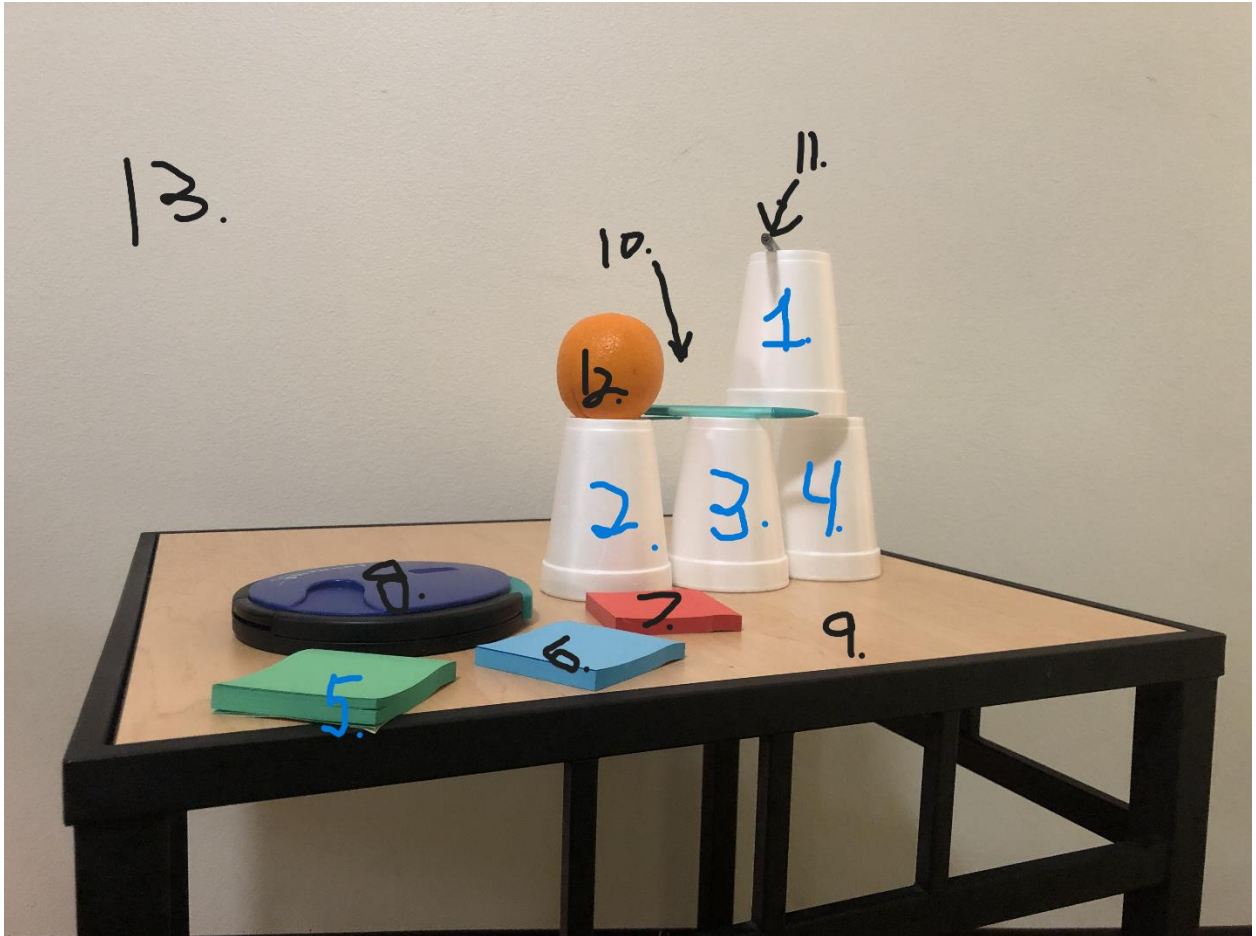
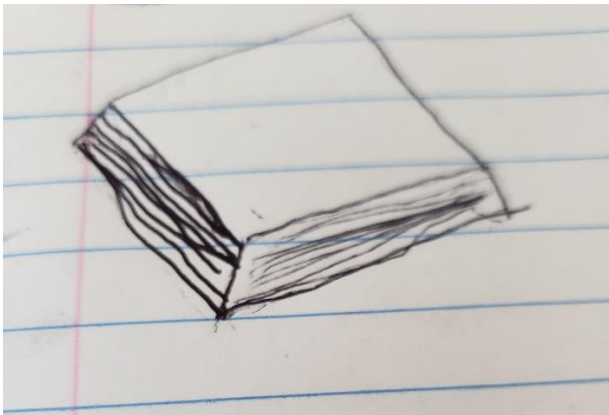**Objects**

1-4. Styrofoam Cups 1 – 4 (foreground)

To Render the cups in openGL we plan to create a cup object that is based of a cone shape and then add materials/shaders to add the unique shadows for each cup.
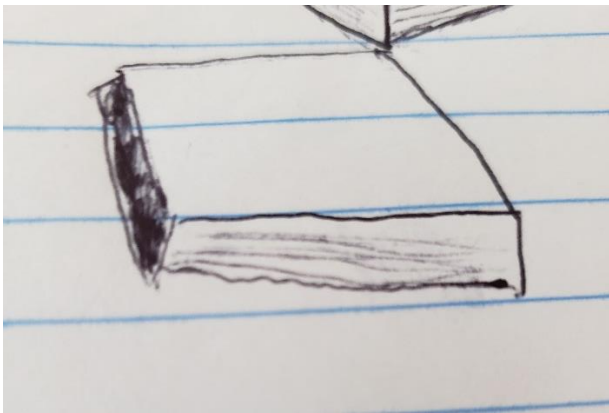
5. **Green Sticky Notes (forground)**

Like the cups, we plan to create a sticky note Object that we can duplicate and adjust and put in the colors and orientations.

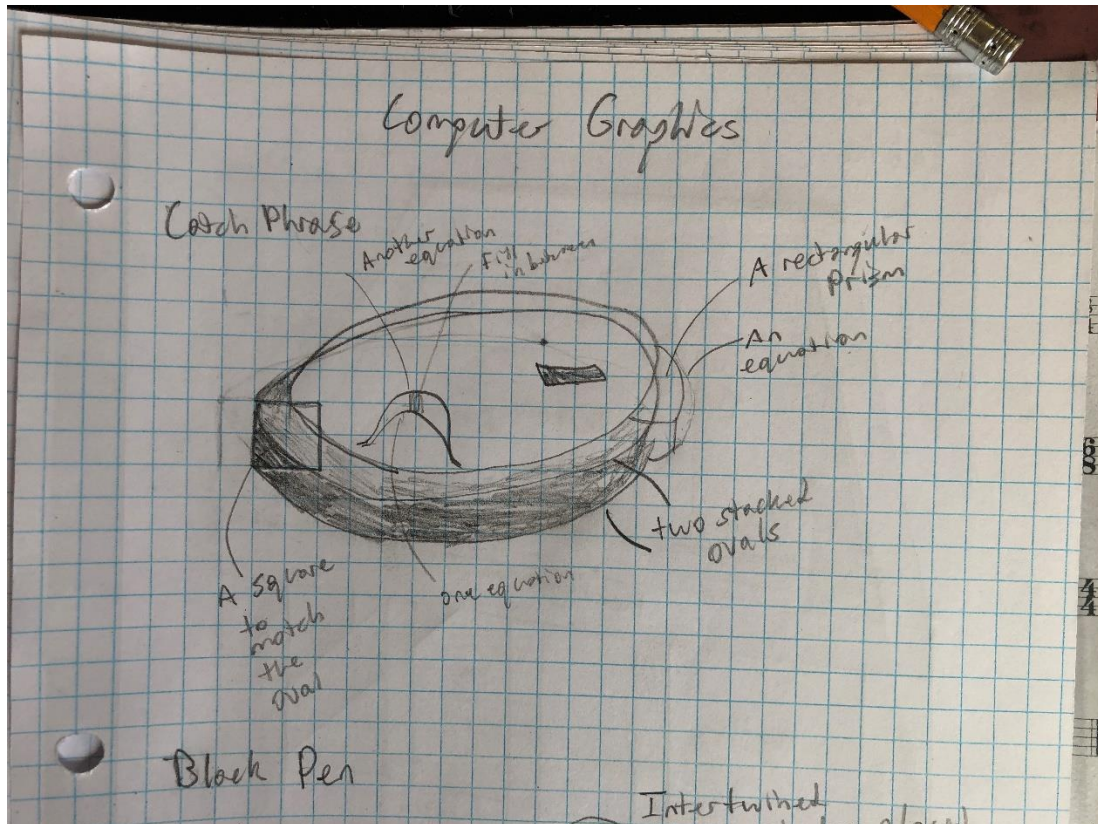6. **Blue Sticky Notes (foreground)**

Like the cups, we plan to create a sticky note Object that we can duplicate and adjust and put in the colors and orientations.

7. **Red Sticky Notes (foreground)**

Like the cups, we plan to create a sticky note Object that we can duplicate and adjust and put in the colors and orientations.

8. **Catch Phrase (Foreground)**

The catch phrase disk can be modeled by three stacked ovals. The bottom oval and the oval above are both matching black. The top oval will be the blue color. All of the ovals will have a rough texture to make it more realistic. In between the bottom and middle oval there will be a square on the left with the same matching black color. This will hide the "gap" the ovals leave. The top of the disc will use Beizer/Mathematical curves and fill the space between the curves.

9. Table (in-between)
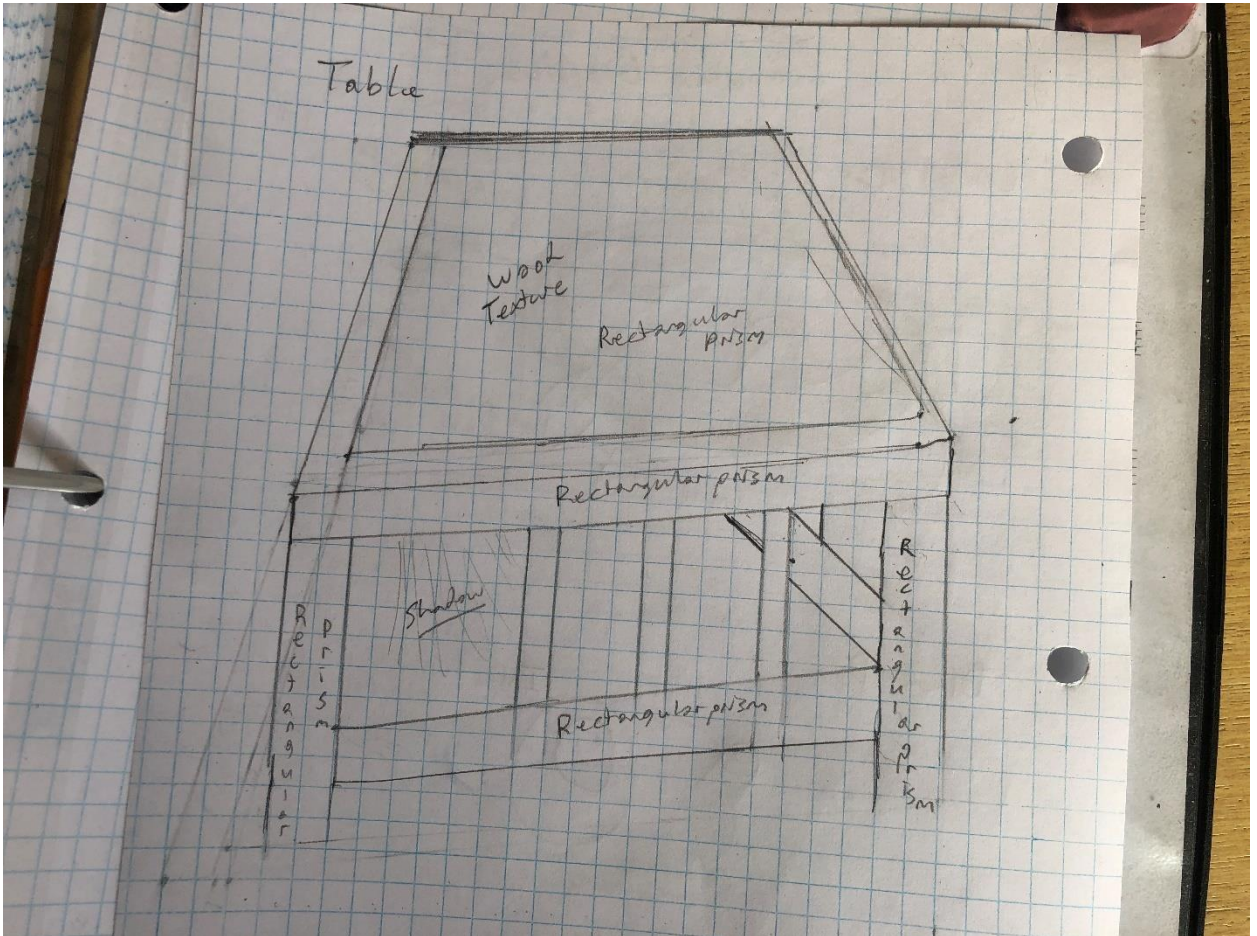
The table is composed of several rectangular prisms attached together. The top features a wooden texture. To render this, each rectangular prism in the table will be separate, but tied together. The top will be two rectangular prisms, the wooden one being "inside" the black outer prism.

10. Green Pen (foreground)
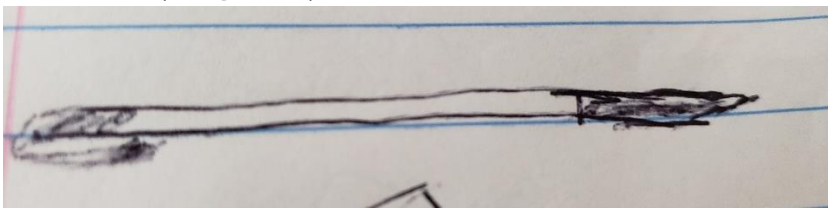
To draw the pen in OpenGL. We plan to first create a cylinder and then add in the unique features that make it look like a pen. Rendering the pen's clip might be especially difficult

11. Black Pen (foreground)
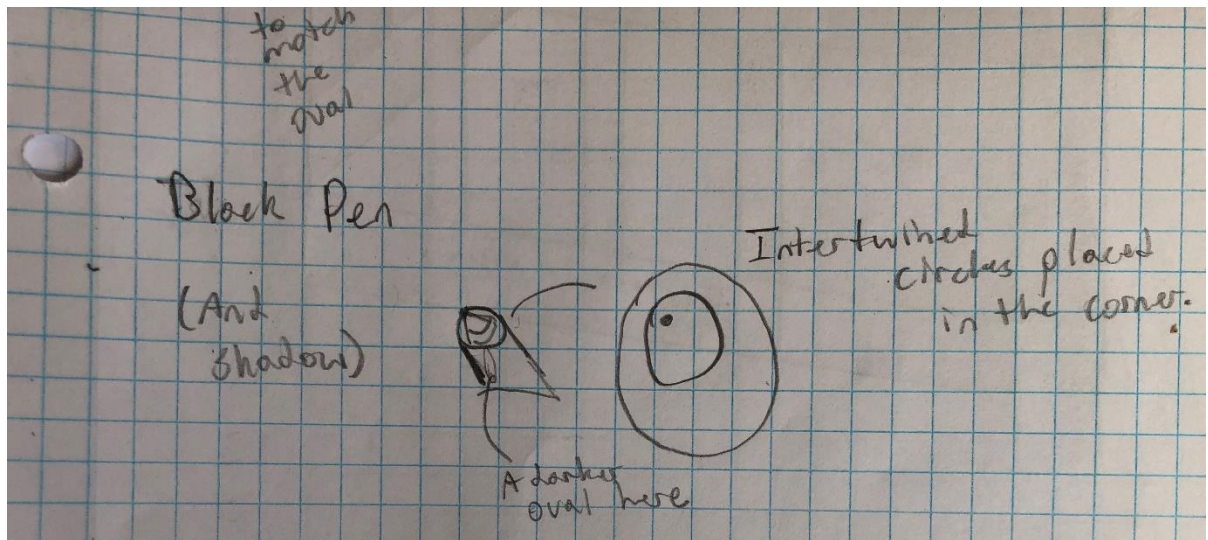
The black pen features a set of intertwined circles placed in the top left corner of each circle. The colors of the circles will be darker colors to match the pen color.

12. Orange (foreground)
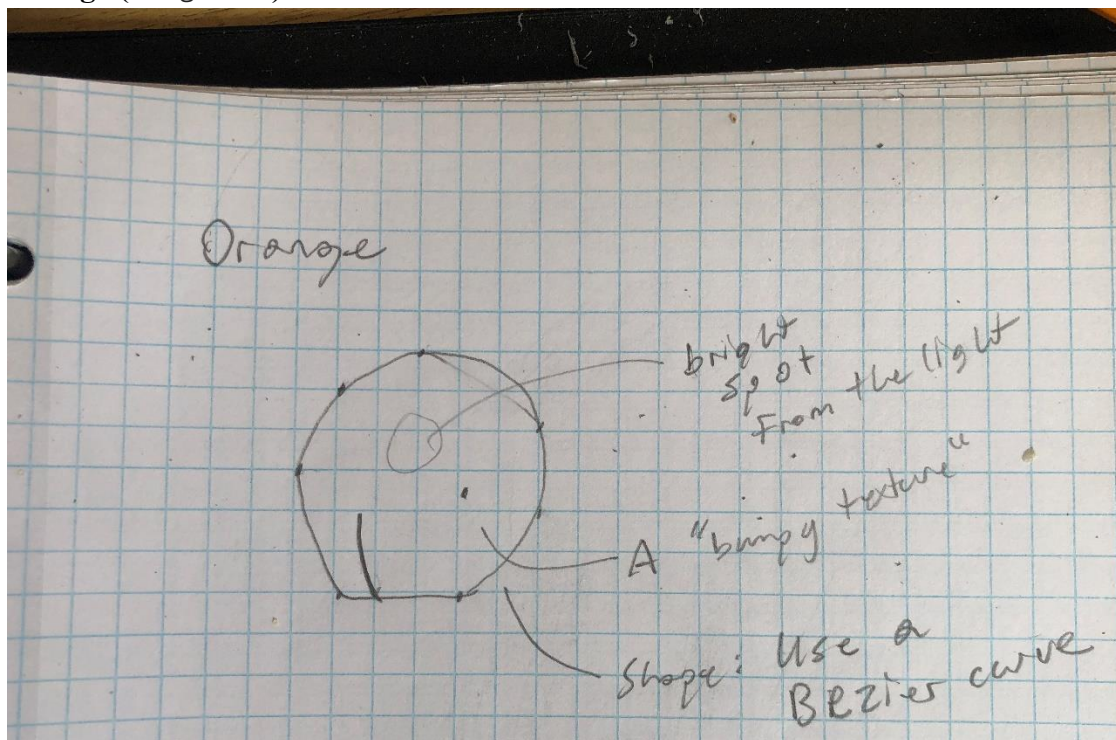
The orange in the image is a circular shape. However, it will be modeled by a many sided shape, to make it more realistic (verses a perfect circle). Then, the shape will have a bumpy texture, and also have a bright spot on it.

13. Wall (background)
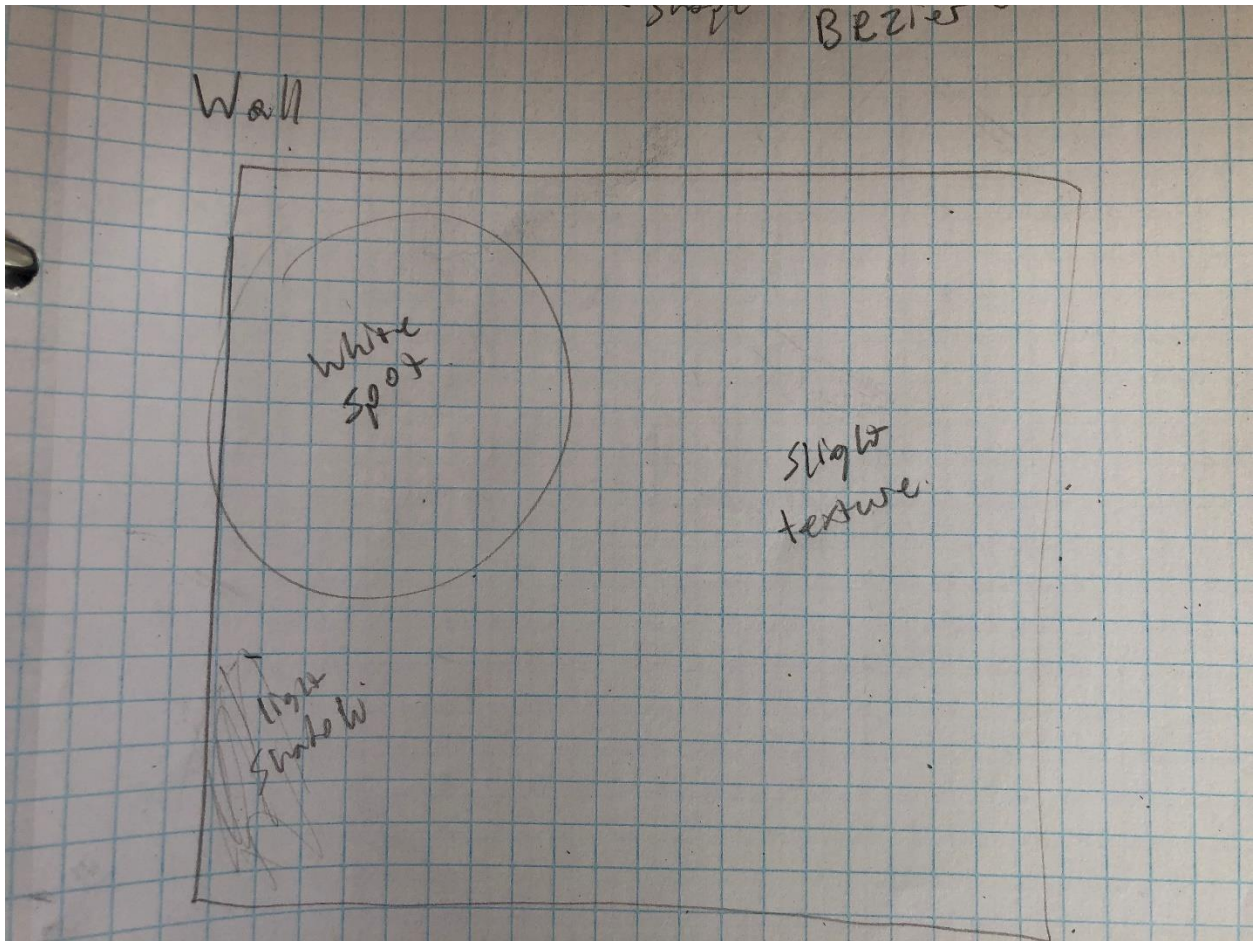
To render a wall, a off-white plane will be created. On it, there will be a white spot from the light. The plane also will have a slight texture to match the stucco on the wall. The wall also will receive shadows from the other objects in the scene.

OpenGL Rendering

Reference Picture:

Final Picture:

GitHub: https://github.com/BenRobotics101/ComputerGraphics

Math Functions in common:

There are several common math functions that are implemented in this project, namely a mapValue function and a MultiPolygon Class.
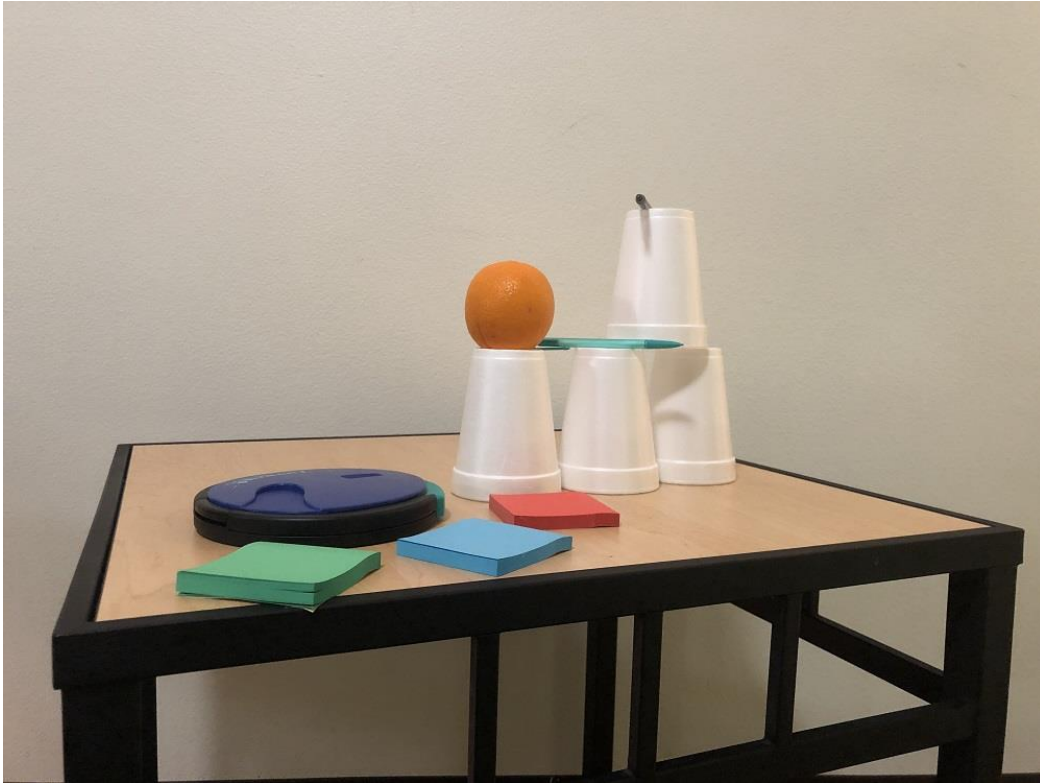
A mapValue function takes a number with a specific range, and "maps" it linearly to another range. For example, the number 5 out of the range 1 through 10. Five is halfway between one and ten. If this number then was mapped to the range of 1 to 100, the result would be 50, as 50 is halfway between 1 and 100. This is extremely useful as one can proportionally convert from one number range to another. In this context, when mixing two colors, one can pass a value resembling a "mix ratio" of 0 to 1. Then, the output range would be the first color and the second color. Then, a number of 0.4 would map to the color that is 40% in between the two colors.

The MultiPolygon class is a "primitive" shape that was created for this project. It takes two polygons and creates additional polygons to connect the original two together.
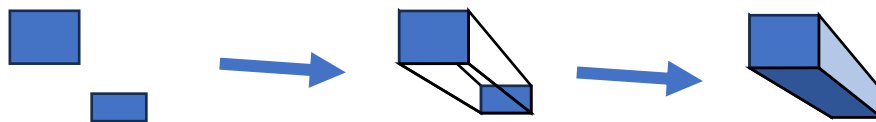
This helps greatly in the "3D illusion" that this project utilizes. See the Camera section for more details. In this project, many of the objects are MultiPolygons. All that is passed in is the boundary points, and the class fills in the rest of the polygons.

Shaders and Shader Concepts

A shader is simply a program with a single function that is run by the GPU. This program receives vector inputs, and outputs a vector. A GPU is specialized to run many parallel computations at once, so if for example 500 vectors were needed to be processed by the same shader (procedure), the GPU would be able to parallel process them. There are two types of shaders. A vector shader and a fragment shader. A vector shader tends to deal with vector transformations, while a fragment shader tends to calculate color values given the vector.

Mathematically, a shader could be thought of as a matrix, where that matrix would be multiplied by a vector to result in a new vector. It is a transformation matrix.

$$N = Av$$

Where N is the new output vector, and A is the transformation matrix (shader) on the input vector v.

An example of this is a shader that rotates an image:

$$N = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} v$$

A shader can be programmed to be that matrix, which the GPU will render, to transform all the vectors parallelly.

In our program, we are not currently using shaders. As the method we decided to use (the 3D illusion method as described below) uses flat 2D primitive shapes to give the illusion of 3D, every item on the screen is currently a simple shape that OpenGL supports out-of-the-box. Later however textures are planned to be added, which would necessitate the need for shaders. Also, there are no "images" that are layered on the project yet. This means that every object drawn is a solid shape with a set color.

In some of the objects, there are for loops that in a way emulate a coloring shader to give gradients onto images.

Textures

For additional realism in the second rendition (project 5), we added a Texture class which loaded an image from the disk. This allows for easy implementation into the other objects. The Texture class also self-scales, where points on the texture can be passed in rather than the default 0.0 to 1.0 scale. Textures are found on the orange, table, catchphrase, wall, cups, and others. Textures were obtained either by taking photos of each object separately (like the wall and table), while others were taken from the reference image itself.

Camera Usage – The 3D illusion

What is a Camera?

A camera is a virtual "eye" placed on the scene that the computer will render for. This is where the world would be transformed via additional matrix transformations to make a 2D image specifically for the 3D "eye".

In this project

To implement this project, we were given a reference image to replicate. In this, instead of using the 3D system of OpenGL, we decided to replicate the image in a 2D manner. That way, exact pixel locations can be recorded as anchor points for the objects in OpenGL. This also gives highly accurate renderings under less time. One disadvantage however is that currently the scene is locked to one perspective.

The 3D illusion.

If we are not using a camera, then how are we getting such a sharp 3D? Well, each object is drawn using polygons that give the illusion of 3D. For instance:
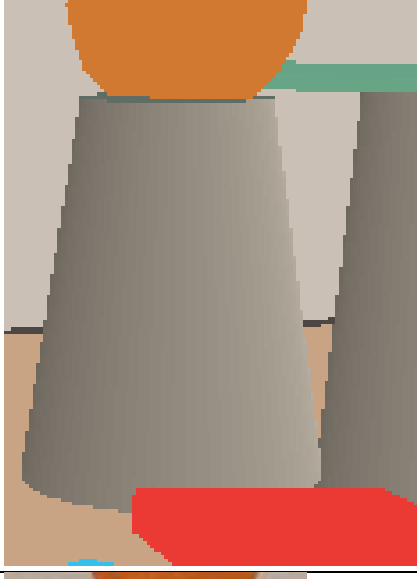
Now, to make something "3D", one only has to make polygons that fit together just right to make a 3D object. This also explains why the "MultiPolygon" class is used frequently. All that is passed into the MultiPolygon class are the two rectangles on the left. Then, it automatically makes additional polygons to make it "3D".
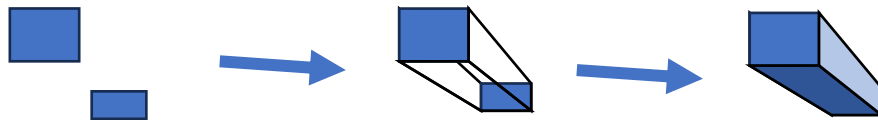
Now, this means we have a "camera". This is seen by using perspective/vantage points where parallel lines in shapes converge on a single point.

Objects specific:

1. **Cup 1**

Reference Image	
First Rendition	
Second Rendition	

For the cups, the shape of the cups was drawn as done in the last version but a new function textureHandler was used to get put an image of the cup on the renduring, the correct cordanats where mapped to the cup so it can be applied right. To get the genernal shape 2 ellipses were drawn. The x and y radius where set and points and the following function was used to draw it

X + radius * cos(θ)

Y + radius * sin(θ)

```
        Point center = {482, 333};
        double radiusX = 34;
        double radiusY = 2;

        int precision = 5;
        // double

        for(int r = 0; r > -180; r -= precision) // Half a
circle, the top (y is flipped.)
        {
            double rad = (PI * r) / 180.0;
            float x = center.x + radiusX * cos(rad);
            float y = center.y + radiusY * sin(rad);


cupTexture.markCoord(Window::mapValue(x,428,534,22,383),Window::
mapValue(y,479.8,333,1,500));
            glVertex2f(x,y);
        }

        radiusX = 53;
        radiusY = 14;
        center = {480, 465};

        for(int r = 180; r > 0; r -= precision) // Half the
circle, the bottom.
        {
            double rad = (PI * r) / 180.0;
            float x = center.x + radiusX * cos(rad);
```

```
            float y = center.y + radiusY * sin(rad);

            //cupTexture.markCoord()

cupTexture.markCoord(Window::mapValue(x,428,534,22,383),Window::
mapValue(y,479.8,333,1,500));
            glVertex2f(x,y);
        }

    glEnd();
    cupTexture.stop();
```

2. Cup 2

Reference Image	
First Rendition	
Second Rendition	

For the cups, the shape of the cups was drawn as done in the last version but a new function textureHandler was used to get put an image of the cup on the renduring, the correct cordanats where mapped to the cup so it can be applied right. To get the genernal shape 2 ellipses were drawn. The x and y radius where set and points and the following function was used to draw it

$X + radius * \cos(\theta)$

$Y + radius * \sin(\theta)$

```
        Point center = {581, 333};
        double radiusX = 34;
        double radiusY = 2;

        int precision = 5;
        // double

        for(int r = 0; r > -180; r -= precision) // Half a
circle, the top (y is flipped.)
        {
            double rad = (PI * r) / 180.0;
            float x = center.x + radiusX * cos(rad);
            float y = center.y + radiusY * sin(rad);


cupTexture.markCoord(Window::mapValue(x,531,648,76,426),Window::
mapValue(y,463,315,74,474));
            glVertex2f(x,y);
        }

        radiusX = 53;
        radiusY = 14;
        center = {581, 471};

        for(int r = 180; r > 0; r -= precision) // Half the
circle, the bottom.
        {
            double rad = (PI * r) / 180.0;
            float x = center.x + radiusX * cos(rad);
            float y = center.y + radiusY * sin(rad);
```

```
            //cupTexture.markCoord()

cupTexture.markCoord(Window::mapValue(x,531,648,76,426),Window::
mapValue(y,463,315,74,474));
            glVertex2f(x,y);
        }

    glEnd();
    cupTexture.stop();
```
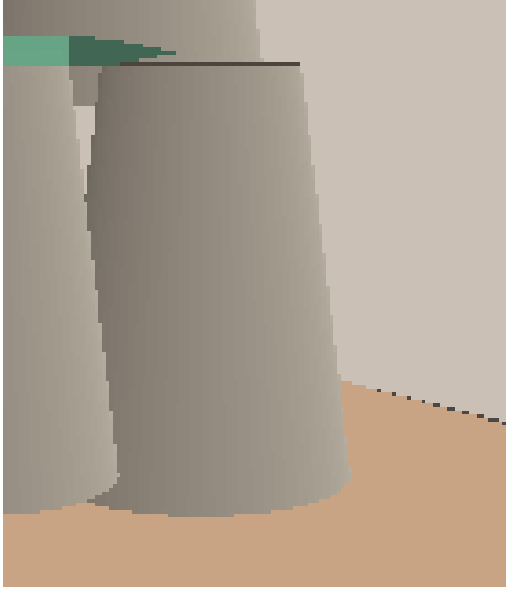
3. **Cup 3**

Reference Image	

First Rendition	
Second Rendition	

For the cups, the shape of the cups was drawn as done in the last version but a new function textureHandler was used to get put an image of the cup on the renduring, the correct cordanats where mapped to the cup so it can be applied right. To get the genernal shape 2 ellipses were drawn. The x and y radius where set and points and the following function was used to draw it

X + radius * cos(θ)

Y + radius * sin(θ)

```cpp
        Point center = {581, 333};
        double radiusX = 34;
        double radiusY = 2;

        int precision = 5;
        // double

        for(int r = 0; r > -180; r -= precision) // Half a
circle, the top (y is flipped.)
        {
            double rad = (PI * r) / 180.0;
            float x = center.x + radiusX * cos(rad);
            float y = center.y + radiusY * sin(rad);


cupTexture.markCoord(Window::mapValue(x,531,648,76,426),Window::
mapValue(y,463,315,74,474));
            glVertex2f(x,y);
        }

        radiusX = 53;
        radiusY = 14;
        center = {581, 471};

        for(int r = 180; r > 0; r -= precision) // Half the
circle, the bottom.
        {
            double rad = (PI * r) / 180.0;
            float x = center.x + radiusX * cos(rad);
            float y = center.y + radiusY * sin(rad);

            //cupTexture.markCoord()

cupTexture.markCoord(Window::mapValue(x,531,648,76,426),Window::
mapValue(y,463,315,74,474));
            glVertex2f(x,y);
        }

    glEnd();
    cupTexture.stop();
```
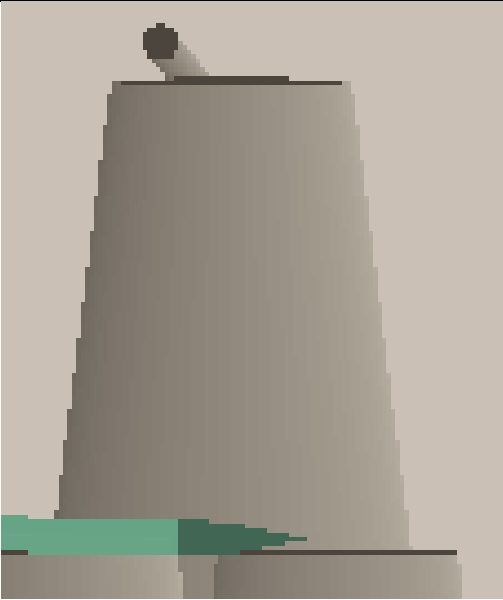

4. Cup 4

Reference Image	
First Rendition	

| Second Rendition | |

For the cups, the shape of the cups was drawn as done in the last version but a new function textureHandler was used to get put an image of the cup on the renduring, the correct cordanats where mapped to the cup so it can be applied right. To get the genernal shape 2 ellipses were drawn. The x and y radius where set and points and the following function was used to draw it

X + radius * cos(θ)

Y + radius * sin(θ)

```
        Point center = {581, 333};
        double radiusX = 34;
        double radiusY = 2;

        int precision = 5;
        // double

        for(int r = 0; r > -180; r -= precision) // Half a
circle, the top (y is flipped.)
        {
            double rad = (PI * r) / 180.0;
            float x = center.x + radiusX * cos(rad);
            float y = center.y + radiusY * sin(rad);


cupTexture.markCoord(Window::mapValue(x,531,648,76,426),Window::
mapValue(y,463,315,74,474));
```

```
            glVertex2f(x,y);
        }

        radiusX = 53;
        radiusY = 14;
        center = {581, 471};

        for(int r = 180; r > 0; r -= precision) // Half the
circle, the bottom.
        {
            double rad = (PI * r) / 180.0;
            float x = center.x + radiusX * cos(rad);
            float y = center.y + radiusY * sin(rad);

            //cupTexture.markCoord()

cupTexture.markCoord(Window::mapValue(x,531,648,76,426),Window::
mapValue(y,463,315,74,474));
            glVertex2f(x,y);
        }

    glEnd();
    cupTexture.stop();
```

5. Green sticky notes

Reference Image	
First Rendition	
Second Rendition	

To create the Sticky Notes, two simple faces were created and the MultiPologon was used to create the rendering.

MultiPolygon gives the 3D effect (operates as our 3D Primitive).

The change between the first rendition and second rendition is that colors were matched to the original image to make it more realistic.

```
// green
    Point greenTop[4] = {
        {360, 528},
        {298, 557},
        {167, 546},
        {243, 517},
    };
    AnchorFace greenTopFace = AnchorFace(greenTop, 4);
    greenTopFace.setColor(120,183,138);

    Point greenTopLower[4] = {
        {362, 542},
        {297, 578},
        {165, 566},
        {248, 528}
```
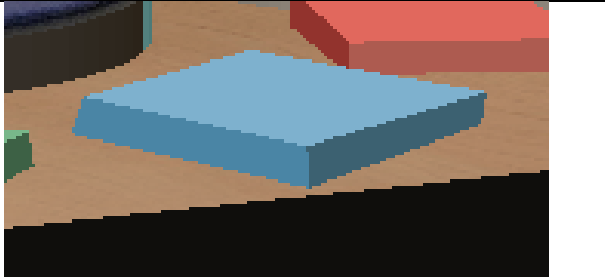
```
    };
    AnchorFace greenTopLowFace = AnchorFace(greenTopLower, 4);
    greenTopLowFace.setColor(79,147,98);

    MultiPolygon green = MultiPolygon(&greenTopFace,
&greenTopLowFace);
    Face** walls = green.getColorFaces();

    //walls = green.getColorFaces();
    for(int i = 0; i < green.getColorFacesNum(); i++)
    {
        if( i % 2 != 0)
        {
            walls[i]->setColor(65,128,99);
        }
        else
        {
            walls[i]->setColor(61,97,69);
        }
    }
```

6. Blue sticky notes

Reference Image	
First Rendition	
Second Rendition	

To create the Sticky Notes, two simple faces were created and the MultiPologon was used to create the rendering.
MultiPolygon gives the 3D effect (operates as our 3D Primitive).

The change between the first rendition and second rendition is that colors were matched to the original image to make it more realistic.

```
Point blueTop[4] = {
        {546, 512},
        {473, 534},
        {382, 514},
        {450, 495}
    };
    AnchorFace blueTopFace = AnchorFace(blueTop, 4);
    blueTopFace.setColor(126, 177, 206);

    Point blueTopLower[4] = {
        {544, 525},
        {474, 551},
```

```
        {377, 529},
        {452, 505}
    };
    AnchorFace blueTopLowFace = AnchorFace(blueTopLower, 4);
    blueTopLowFace.setColor(52, 192, 235);

    MultiPolygon blue = MultiPolygon(&blueTopFace,
&blueTopLowFace);

    walls = blue.getColorFaces();
    for(int i = 0; i < blue.getColorFacesNum(); i++)
    {
        if( i % 2 == 0)
        {
            walls[i]->setColor(60,97,113);
        }
        else
        {
            walls[i]->setColor(74,133,165);
        }
    }
```

7. Red sticky notes

Reference Image	
First Rendition	
Second Rendition	

To create the Sticky Notes, two simple faces were created and the MultiPologon was used to create the rendering.
MultiPolygon gives the 3D effect (operates as our 3D Primitive).

The change between the first rendition and second rendition is that colors were matched to the original image to make it more realistic.

```
// red
Point redTop[4] = {
        {591, 490},
        {490, 492},
        {467, 470},
        {554, 470}
    };
    AnchorFace redTopFace = AnchorFace(redTop, 4);
    redTopFace.setColor(225,104,93);
```

```
    Point redTopLower[4] = {
        {591, 503},
        {490, 505},
        {466, 485},
        {550, 476}
    };
    AnchorFace redTopLowFace = AnchorFace(redTopLower, 4);
    redTopLowFace.setColor(225,104,93);

    MultiPolygon red = MultiPolygon(&redTopFace,
&redTopLowFace);

    walls = red.getColorFaces();
    for(int i = 0; i < red.getColorFacesNum(); i++)
    {
        if( i % 2 != 0)
        {
            walls[i]->setColor(146,51,45);
        }
        else
        {
            walls[i]->setColor(173,91,80);
        }
    }
```

8. Catch Phrase game

Original	
First Rendition	
Second Rendition	

For the catch phrase game, two ellipses were created, with the parametric equations. The elipses were fed into the MultiPolygon class. Then, each wall (generated polygon) goes through a for loop for coloring.

Also, to rotate the catchphrase game, the mapValue function was used. The input range is the x value between the x bounds. Then, this is converted to a y transform value which is multiplied by 4.75 to create the rotation/skewing.

There is an if statement to change the coloring from the green to black.

MultiPolygon gives the 3D effect (operates as our 3D Primitive)

In the second rendition, a texture was added to the top ellipse to model the catchphrase game. The MapValue function was used to translate the physical coordinates to the texture coordinates.

```
Point center = {303, 469};
    double radius = 5;

    int precision = 5;
    Point circle[360 / precision];
    Point circleTx[360 / precision];
    // double
    int counter = 0;
    for(int r = 0; r < 360; r += precision) // 15deg precision.
    {
        double rad = (PI * r) / 180.0;
        circle[counter] = {center.x + 110 * cos(rad), center.y +
32 * sin(rad)};

        circle[counter].x =
Window::mapValue(circle[counter].x,193.0f,413.0f,180.0f,410.0f);
// scaling and rotation
        circle[counter].y =
Window::mapValue(circle[counter].y,437.0f,501.0f,450.0f,495.0f);

        circle[counter].y = circle[counter].y - 4.75f *
Window::mapValue(circle[counter].x,180.0f,410.0f,-1.0f,1.0f);

        circleTx[counter].x =
Window::mapValue(circle[counter].x,180.0f,410.0f,40.0f,960.0f);
        circleTx[counter].y =
Window::mapValue(circle[counter].y,450.0f,495.0f,581.0f,400.0f);

        std::cout << "CIRCLE" << circle[counter].x << ' '  <<
circle[counter].y << '\n';
        counter++;
    }

    center.y = center.y + 32;
```

```cpp
    Point circle2[360 / precision];
    // double
    counter = 0;
    for(int r = 0; r < 360; r += precision) // 15deg precision.
    {
        double rad = (PI * r) / 180.0;
        circle2[counter] = {center.x + 110 * cos(rad), center.y
+ 32 * sin(rad)};

        circle2[counter].x =
Window::mapValue(circle2[counter].x,193.0f,413.0f,180.0f,410.0f)
;
        circle2[counter].y =
Window::mapValue(circle2[counter].y,437.0f,501.0f,450.0f,495.0f)
;

        circle2[counter].y = circle2[counter].y - 4.75f *
Window::mapValue(circle2[counter].x,180.0f,410.0f,-1.0f,1.0f);

        counter++;
    }

    AnchorFace top = AnchorFace(circle, 360 / precision);
  //  top.setColor(84, 87, 120);
    top.setColor(255, 255, 255);
    top.setTexture(&topTexture, circleTx);

    AnchorFace bottom = AnchorFace(circle2, 360 / precision);
    bottom.setColor(11,11,9);
    MultiPolygon pen = MultiPolygon(&top, &bottom);
    Face** walls = pen.getColorFaces();
    for(int x = 0; x < pen.getColorFacesNum(); x++)
    {
        float grad = Window::mapValue((float)((x - 1) %
pen.getColorFacesNum()) , 0.0f, (float)pen.getColorFacesNum(),
0.0f, 2.0f);
        float color[3];
        if(grad < 0.5f)
        {
            Window::mixColor(color, grad, 41,33,22,82,83,85);
```
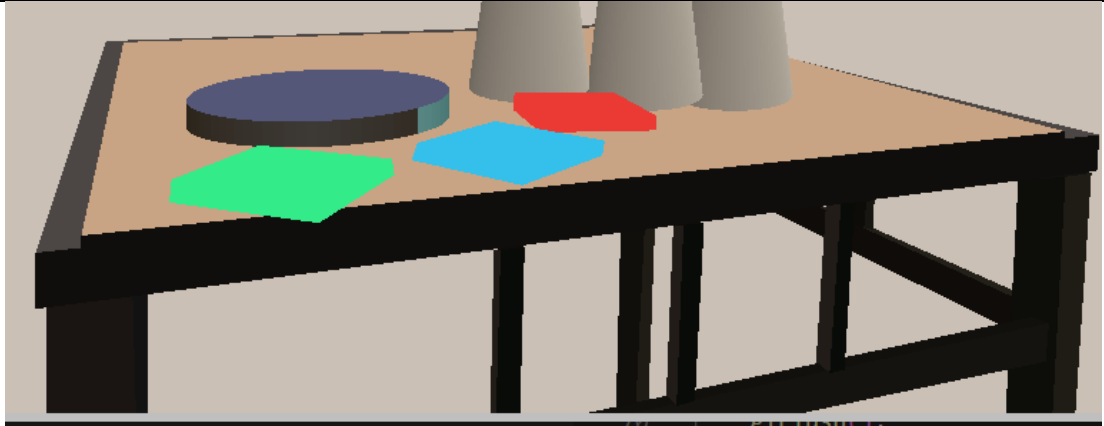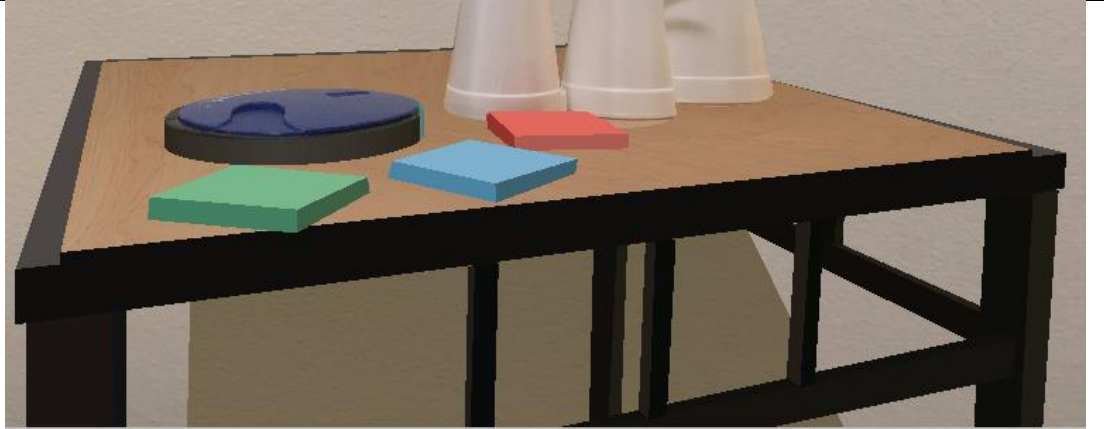
```
        }
        else
        {
            Window::mixColor(color, 1.0 - grad, 41,33,22,
82,83,85);
        }
        if(x < 17 / precision )
        {
            Window::mixColor(color, (float)((float)x / (17 /
precision)),70.0f,116.0f,113.0f, 90.0f, 136.0f, 133.0f);
        }
        walls[x]->setColor((uint8_t)color[0],(uint8_t)color[1],
(uint8_t)color[2]);
    }

    pen.draw();
```

9. Table

Original	
First Rendition	
Second Rendition	

The Table object was the most complicated object to replicate. The table really just consisted of many MultiPolygon objects. The table was broken down into the following shapes (in render order):
1. Wood top
2. Black top (the border)

3. The left-most leg
4. The left-most decorative support
5. The middle decorative support
6. The right decorative support
7. The crossbar most forward
8. The right leg
9. The crossbar going back into the wall
10. The support connecting #9 above to the black top
11. The leg in the far corner

In creating this, a graphical editing application was used to extract the exact pixel coordinates of each object. Then, those objects were passed into the MultiPolygon class. Then, the colors were matched to the image for realism.

MultiPolygon gives the 3D effect (operates as our 3D Primitive)

The second rendition of the table included adding the wood texture to the top, and also adding a table shadow (located in wall.cpp).

```
See the Code in Table.cpp

Is very long (11 shapes)
```

10. Green Pen

Original	
First Rendition	

| Second Rendition | |

The Pen was created in 2 main parts, the shaft of the pen, and the point. To create shaft, 2 ellipses were drawn and connected with the MultiPolygon. For the point of the pen several small rectangles of decreasing size where created to match the shape of the point.

MultiPolygon gives the 3D effect (operates as our 3D Primitive)

```
Point center = {615, 328};
    double radiusX = 2;
    double radiusY = 5;

    int precision = 5;
    Point circle[360 / precision];
    // double
    int counter = 0;
    for(int r = 0; r < 360; r += precision) // 15deg precision.
    {
        double rad = (PI * r) / 180.0;
        circle[counter] = {
            center.x + radiusX * cos(rad),
            center.y + radiusY * sin(rad)
        };
        //
        std::cout << "Ellipse" << circle[counter].x << ' '  <<
circle[counter].y << '\n';
        counter++;
    }
    radiusX = 2;
```

```cpp
    radiusY = 5;
    center = {500, 325};
    Point circle2[360 / precision];
    // double
    counter = 0;
    for(int r = 0; r < 360; r += precision) // 15deg precision.
    {
        double rad = (PI * r) / 180.0;
        //circle2[counter] = {center.x + cos(rad) * radius,
center.y + sin(rad) * radius};
        circle2[counter] = {
            center.x + radiusX * cos(rad),
            center.y + radiusY * sin(rad)
        };
        counter++;
    }


    AnchorFace top = AnchorFace(circle, 360 / precision);
    top.setColor(103, 163, 132);
    AnchorFace bottom = AnchorFace(circle2, 360 / precision);
    bottom.setColor(103, 163, 132);
    MultiPolygon pen = MultiPolygon(&top, &bottom);
    Face** walls = pen.getColorFaces();
    for(int x = 0; x < pen.getColorFacesNum(); x++)
    {
        float grad = Window::mapValue((float)x, 0.0f,
(float)pen.getColorFacesNum(), 1.0f, 0.0f);
        float color[3];
        Window::mixColor(color, grad, 103, 163, 132,105, 165,
135);
        walls[x]->setColor((uint8_t)color[0],(uint8_t)color[1],
(uint8_t)color[2]);
    }


    Point redTop[4] = {
        {615, 323},
        {623, 323},
        {632, 324},
        //{642, 325},
```

```
        //{647, 328},
        {652, 328}
    };
    AnchorFace redTopFace = AnchorFace(redTop, 4);
    redTopFace.setColor(65, 102, 83);

    Point redTopLower[4] = {
        {615, 333},
        {623, 332},
        {632, 332},
        {632, 332},

//        {642, 331},
      // {647, 328},
       //{650, 328}
    };
    AnchorFace redTopLowFace = AnchorFace(redTopLower, 4);
    redTopLowFace.setColor(65, 102, 83);

    MultiPolygon red = MultiPolygon(&redTopFace,
&redTopLowFace);

    walls = red.getColorFaces();
    for(int i = 0; i < red.getColorFacesNum(); i++)
    {
        walls[i]->setColor(65, 102, 83);
    }
```
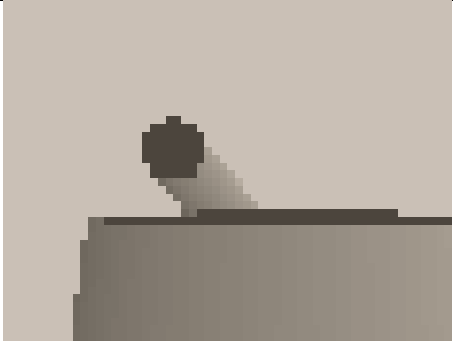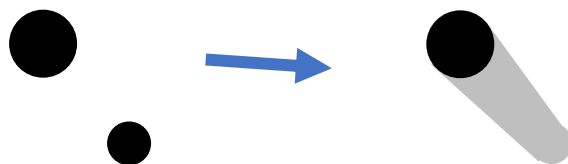
11. Black Pen

Reference Image	

First Rendition	
Second Rendition	

To make the black pen, two circles are drawn as the primary faces. This is fed into the MultiPolygon class.

A mapValue (mixColor) function provides the slight gradient on the pen, mixing the colors as the angle changes on the circle.

MultiPolygon gives the 3D effect (operates as our 3D Primitive)

```
Point center = {610, 188};
    double radius = 5;
```

```cpp
    int precision = 5;
    Point circle[360 / precision];
    // double
    int counter = 0;
    for(int r = -45; r < 360 - 45; r += precision) // 15deg
precision.
    {
        double rad = (PI * r) / 180.0;
        circle[counter] = {center.x + cos(rad) * radius,
center.y + sin(rad) * radius};
        std::cout << "CIRCLE" << circle[counter].x << ' '  <<
circle[counter].y << '\n';
        counter++;
    }

    center = {619, 199};
    Point circle2[360 / precision];
    // double
    counter = 0;
    for(int r = -45; r < 360 - 45; r += precision) // 15deg
precision.
    {
        double rad = (PI * r) / 180.0;
        circle2[counter] = {center.x + cos(rad) * radius,
center.y + sin(rad) * radius};
        counter++;
    }

    AnchorFace top = AnchorFace(circle, 360 / precision);
    top.setColor(76, 69, 61);
    AnchorFace bottom = AnchorFace(circle2, 360 / precision);
    bottom.setColor(145,135,125);
    MultiPolygon pen = MultiPolygon(&top, &bottom);
    Face** walls = pen.getColorFaces();
    for(int x = 0; x < pen.getColorFacesNum(); x++)
    {
        float grad = Window::mapValue((float)x, 0.0f,
(float)pen.getColorFacesNum(), 1.0f, 0.0f);
        float color[3];
        Window::mixColor(color, grad, 33,29,22,182,172,160);
```
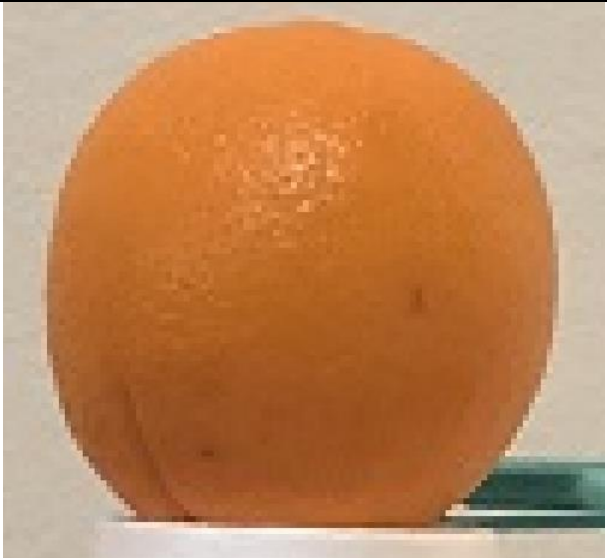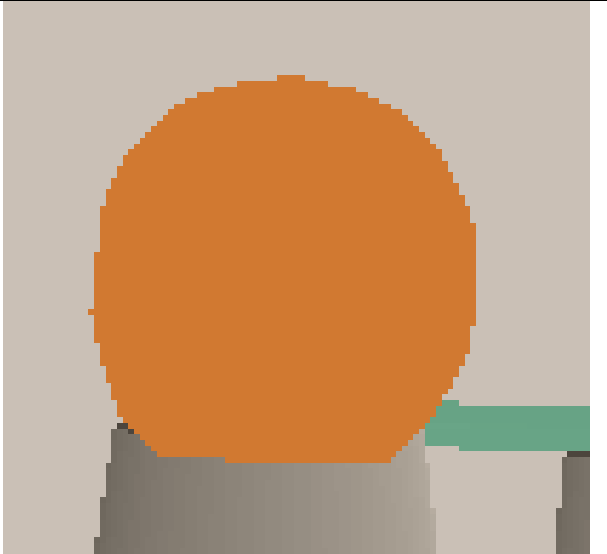
```
        walls[x]->setColor((uint8_t)color[0],(uint8_t)color[1],
(uint8_t)color[2]);
    }
    pen.draw();
```

12. Orange

Reference Image	
First Rendition	

Second Rendition	

For the orange, a simple polygon was created in OpenGL. The orange was viewed on paint, and then points from the orange were plotted into OpenGL. This results in an exact representation of the location of the orange.

In the second rendition, an image of the orange was selected as the texture of this shape for additional realism. The MapValue function was used to translate world space coordinates to texture coordinates.

```
glColor3f(CC(248),CC(208),CC(173));

    // then is verticies

    Point orangePoints[] = {
        {488,249},
        {478,250},
        {469,252},
        {461,256},
        {455,262},
        {450,267},
        {446,275},
        {445,282},
        {444,291},
        {443,301},
        {445,311},
        {449,323},
```

```
                {459,333},
                {467,333},
                {508,334},
                {512,330},
                {517,325},
                {522,317},
                {526,309},
                {528,298},
                {528,284},
                {524,274},
                {518,263},
                {511,257},
                {502,252},
                {492,250},
                {484,262}
        };

        orangeTexture.start();
        glBegin(GL_TRIANGLE_FAN);
        for(int i = 0; i < 27; i++)
        {

orangeTexture.markCoord(Window::mapValue(orangePoints[i].x,443,5
28, 28, 224), Window::mapValue(orangePoints[i].y,417, 248.3, 13,
228));
            glVertex2f(orangePoints[i].x,orangePoints[i].y);
        }
        glEnd();
        orangeTexture.stop();
```
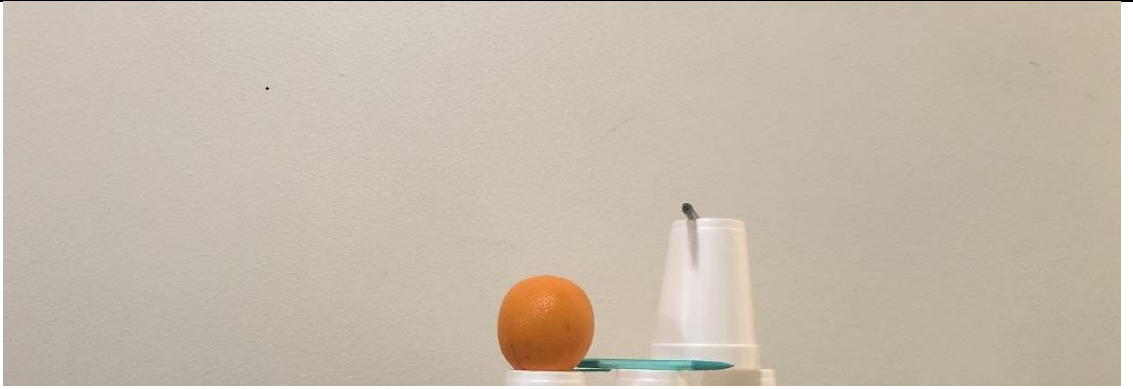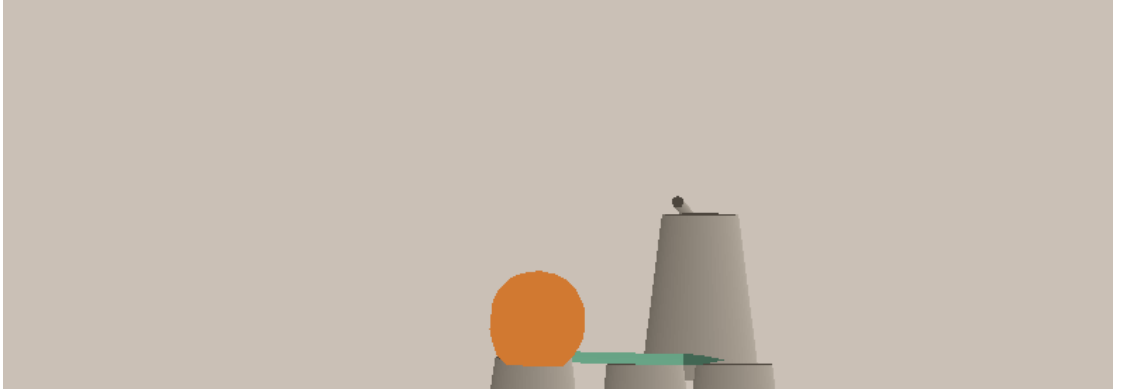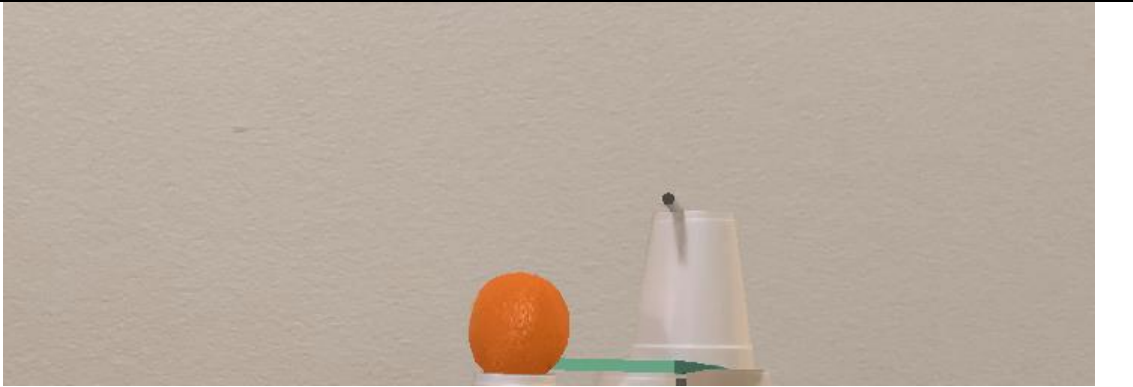
13. Wall

Reference Image	
First Rendition	
Second Rendition	

In the first rendition, the wall is created by simply setting the clear color of the window to the wall color.

In the second rendition, a class was created to add texture support. Also, the table shadow was drawn on the wall as it shared the same texture as the wall. One side effect discovered and used to our advantage is that textures are semi-transparent, allowing for a color base to "bleed through". This allowed for shadowing.

```
glColor3f(CC(140), CC(129), CC(104));
  // glColor3f(CC(202), CC(192), CC(182));
    wallTexture.start();
    glBegin(GL_POLYGON);
```

```cpp
wallTexture.markCoord(Window::mapValue(200,0,1000,1024,0),Window
::mapValue(630,750,0,1024,0));
        glVertex2f(203,630);

wallTexture.markCoord(Window::mapValue(200,0,1000,1024,0),1024);
        glVertex2f(200,750);

wallTexture.markCoord(Window::mapValue(782,0,1000,1024,0),1024);
        glVertex2f(782,750);

wallTexture.markCoord(Window::mapValue(694,0,1000,1024,0),Window
::mapValue(570,750,0,1024,0));
        glVertex2f(694,570);
    glEnd();
    wallTexture.stop();

    // WALL
  //  glColor3f(CC(202), CC(192), CC(182));
    glColor3f(CC(230), CC(225), CC(221));
    wallTexture.start();
    glBegin(GL_POLYGON);
        wallTexture.markCoord(1024,1024);
        glVertex2f(0,0);
        wallTexture.markCoord(0,1024);
        glVertex2f(1000,0);
        wallTexture.markCoord(0,0);
        glVertex2f(1000,750);
        wallTexture.markCoord(1024,0);
        glVertex2f(0,750);
    glEnd();
    wallTexture.stop();
```
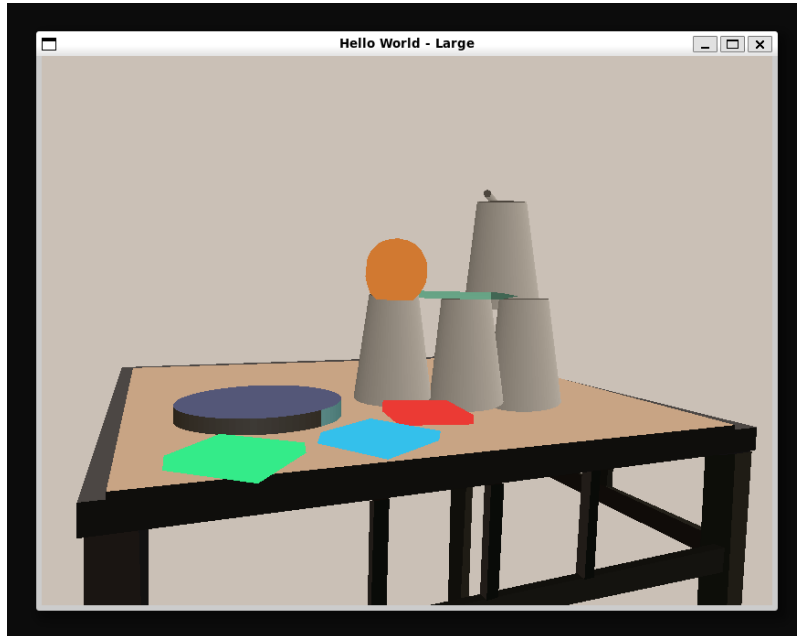
Screenshot of Image

First Rendition

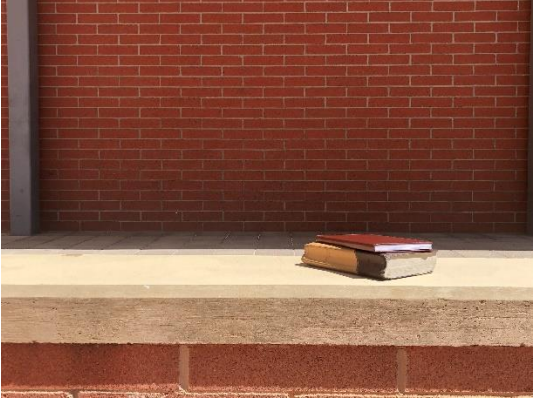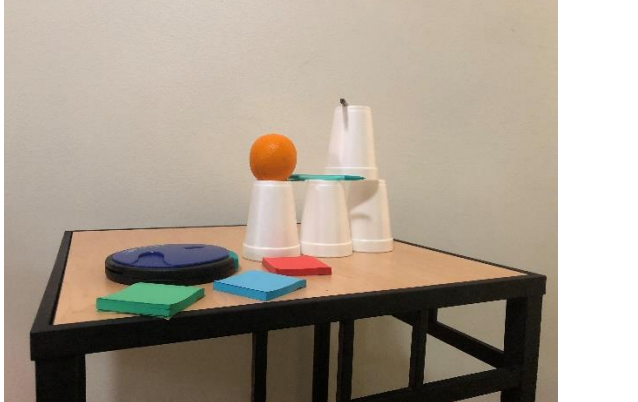### Second Rendition

From Earlier -- Project 2 Documentation

Project Description

The primary goal of this project is to capture real-world photographs for the purpose of developing a fundamental understanding of computer graphics and OpenGL. Six distinct photographs from three different scenes have been taken, and these images will serve as references for future projects, providing students with practical experience in recreating real-world scenes as immersive 3D graphics

Methodology/Approach:

1. **Image Simplicity**: Our approach emphasizes simplicity in the images captured. We deliberately avoid natural textures such as grass or trees and steer clear of complex, highly detailed scenes. This simplification allows for easier translation into graphics and serves as a foundation for learning.
2. **Reflection Avoidance**: Reflections in images can introduce complexities that may be challenging for us to re-create using computer graphics. Thus, we choose scenes and objects that minimize reflective surfaces to keep the initial learning curve manageable.
3. **Visual Interest and Uniqueness**: While keeping simplicity in mind, we also aim for visual interest and uniqueness in our photographs. We actively seek scenes with compelling perspectives, intriguing subjects, or unusual angles that make the images stand out.
4. **Abundance of Objects**: To facilitate future learning and the use of more complex graphical tools, we ensure that our selected scenes contain an ample number of objects. These objects will serve as building blocks for more intricate graphic projects, allowing students to gradually advance their skills.
5. **Practicality**: We balance the quest for unique perspectives with practicality. Scenes are chosen with the understanding that they should be reasonably replicable in later projects. This ensures that the foundational knowledge gained in this stage can be directly applied in future, more complex graphic endeavors.

Pictures

Scene	Picture 1	Picture 2
Desktop items		
Brick wall with book		
random things put on a table		

Statement of work

We, Josh Canode and Benjamin Carter, certify that we alone took these photos with personally owned equipment and that these photos did not come from online sources.