Project 5: Render Your Scene with Primitives

Caleb Klinger, Kyungchan Im (Chris)

Grand Canyon University

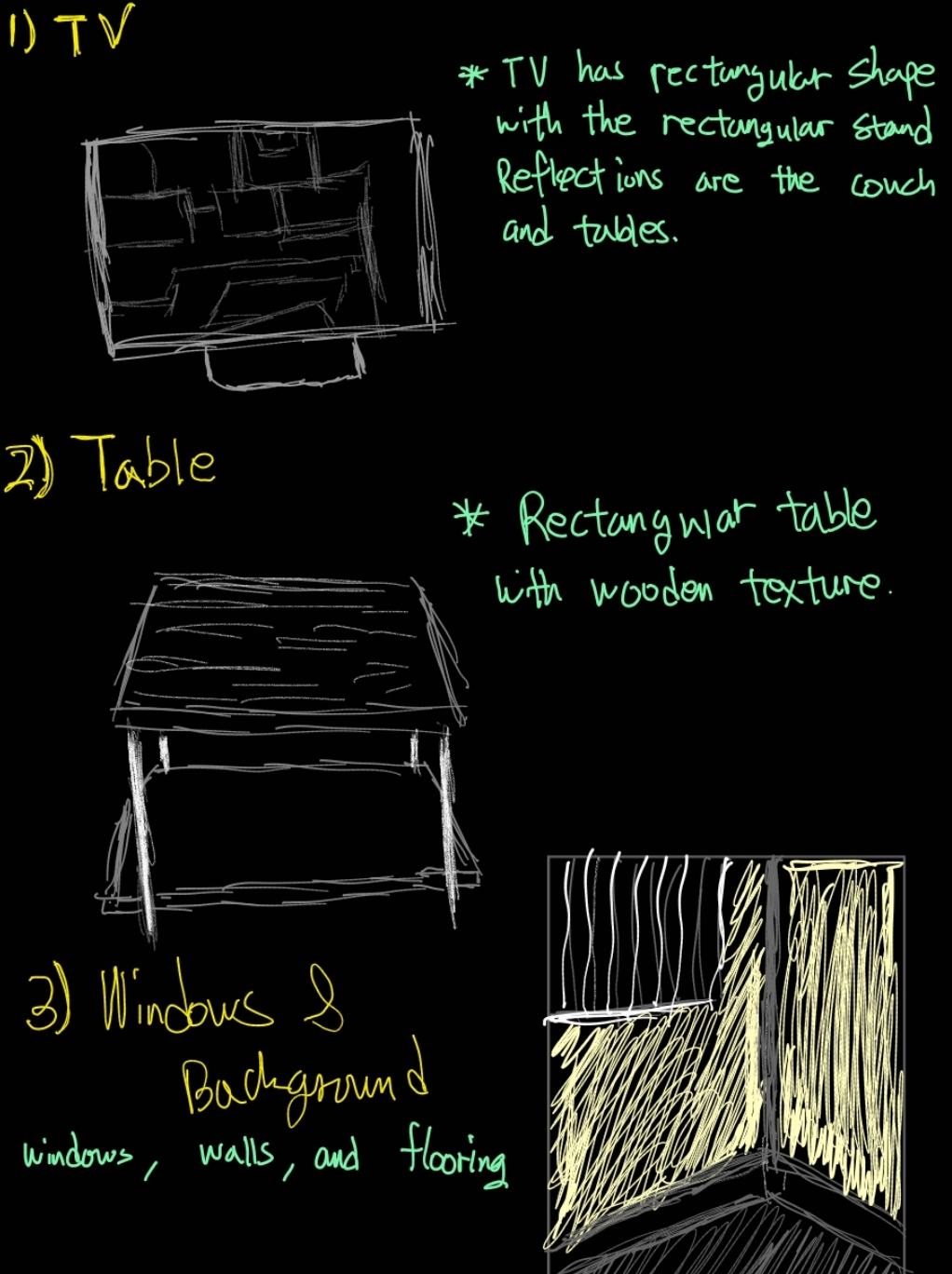
CST - 435 : Computer Graphics Lecture & Lab

Ricardo Citro

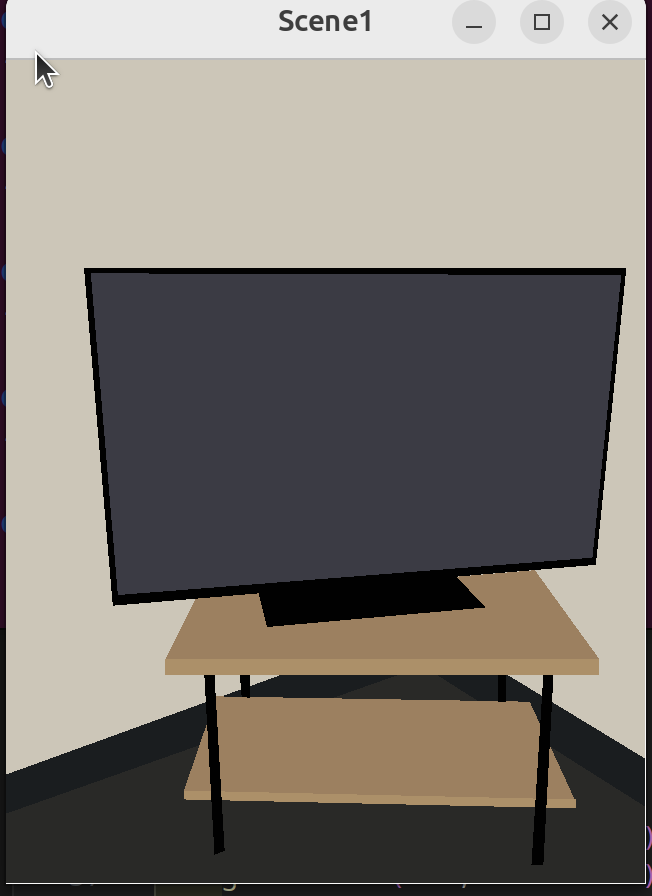
October 15, 2023

**Comparisons**

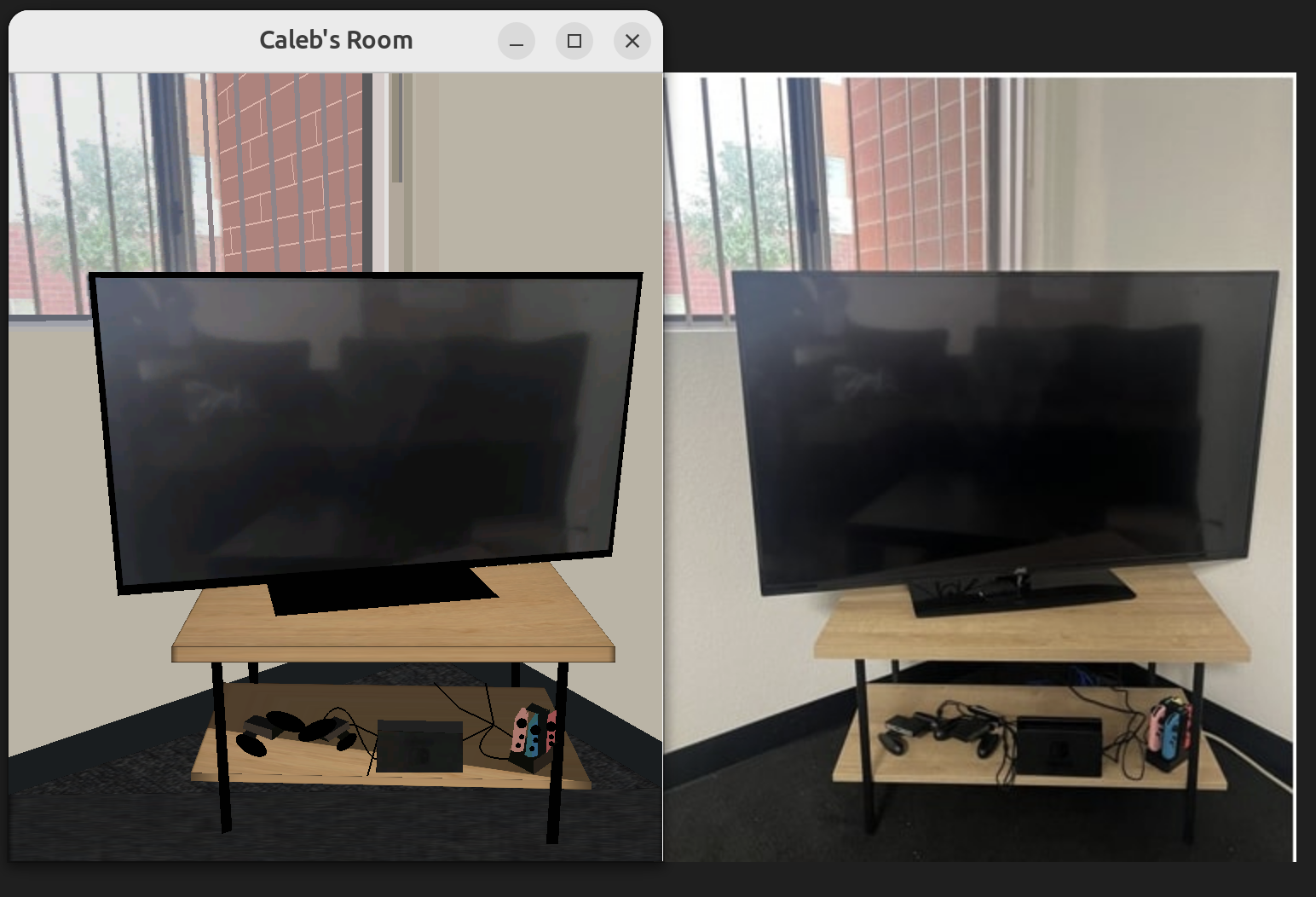
Hand-drawn scene created in Project 3 and reproduce each object using primitives.

Hand Drawn and Real Scene   
 

Transition to more complex meshes, shapes and textures.



Final compared to Original



**Improvements and Primitives**

The primitives we used consisted of polygons and lines.  
GL\_POLYGON: This primitive is used to draw polygons, which are flat, closed shapes bounded by three or more line segments. In your code, GL\_POLYGON is used to draw various parts of the window, the wall, the floor, the table, and the TV. Each glVertex2i(x, y) call specifies a vertex of the polygon, and the vertices are connected in the order they are called.

GL\_LINES: This primitive is used to draw a series of unconnected line segments. In your code, GL\_LINES is used to draw the window blinds. Each pair of glVertex2i(x, y) calls specifies a line segment.

Added Texture Loading with SOIL: We are using the Simple OpenGL Image Library (SOIL) to handle texture loading. SOIL simplifies the process of loading images and creating OpenGL textures. Four textures have been loaded using the SOIL\_load\_OGL\_texture function. These textures represent different materials or objects, namely a wooden texture, carpet, a TV screen, and another texture, related to the Nintendo design.

Error Handling: After each texture is loaded, there's a check to ensure that the loading process was successful. If it was not, an error message prints to the console.

Texture Application:

For drawing objects with textures, you've enabled the texture mapping with glEnable(GL\_TEXTURE\_2D). We binded the appropriate texture with glBindTexture(GL\_TEXTURE\_2D, textureID), where textureID is the OpenGL-assigned ID of the loaded texture. In the given code snippet, you've shown the application of a texture for a window frame using a quad. Texture coordinates (glTexCoord2f) are specified alongside vertex coordinates (glVertex2i) to map parts of the texture to the quad.

The main enhancements are on adding more visual detail and realism to the OpenGL scene by applying different textures to objects. Texture mapping is a powerful technique to elevate the visual fidelity of 3D and 2D applications. By mapping images (textures) onto geometric shapes, you can simulate the appearance of more complex surfaces and materials without increasing the geometric complexity of your scene.

**Use of Meshes, Complex Objects and increasing realism.**

We included grid meshes, Complex lines, added textures, added shadders for shadows   
  
Use of Meshes:

Meshes are essentially collections of vertices, edges, and faces that define the shape of 3D objects in space. A grid mesh, which is a regularly spaced grid of vertices, is particularly useful for representing terrain, floors, or any surface that needs to interact dynamically with other objects (like for collision detection). Using detailed meshes can significantly enhance the realism of a scene. For instance, rather than using a simple quad (rectangle) to represent the ground, a detailed mesh can simulate an uneven terrain. The grid meshes we included helped also help wrap the texture we used.

Complex Objects:

We included more complex objects that have intricate details and curves. In our code, we have utilized a Bézier curve to depict what appears to be a cable or some sort of connector. This curve provides a smooth, realistic appearance, improving upon what might have otherwise been a straight or crudely curved line.

Texture Mapping:

Textures give objects a detailed appearance without the need for more geometry. It's like wrapping a 2D image around a 3D shape. The addition of textures, such as wood, carpet, or images on screens, contributes to the immersion and believability of the scene.

Shaders for Shadows:

Shadows are a crucial element for enhancing realism. They provide depth and context to a scene, allowing objects to feel grounded in their environment. In the code a shadow is simulated using a polygon with a specific color that represents the shadow. We also included a SOIL library to include better textures and dynamic shaders.

Complex Lines and Bézier Curves:

The second part of our code uses a Bézier curve (GL\_LINE\_STRIP) to depict a more intricate line structure, presumably representing a cable or rope. Using mathematical curves like Bézier allows for the creation of smooth, natural-looking curves that can be controlled using a few control points. This approach is more realistic than manually placing every point of the line.

**Mathematical characteristics**

Bézier Curves:

The section of your code that deals with the cable uses a quadratic Bézier curve. A quadratic Bézier curve is defined by three points: a start point, an end point, and a control point. The curve's shape is determined by the position of these three points.

Mathematically, the formula for a quadratic Bézier curve B(t) is:

B(t)=(1−t)2×P0+2(1−t)t×P1+t2×P2

where P0P0, P1P1, and P2P2 are the start, control, and end points respectively, and tt is a parameter that ranges from 0 to 1.

Polygons and Vertex Definitions:

The creation of shapes, such as the shadow polygon, involves defining vertices in 2D space. These vertices, when connected, form the desired shape. Defining the vertices involves determining their x and y coordinates based on the desired shape and size of the polygon.

Depth Testing:

The clearing of the depth buffer (GL\_DEPTH\_BUFFER\_BIT) hints at the use of depth testing in your project. Depth testing ensures that objects closer to the viewer obscure objects that are further away. Mathematically, this involves comparing the depth values of pixels to determine which ones should be displayed.

Coordinate System:

OpenGL uses a coordinate system to position objects in the space (2D or 3D). In your 2D scene:

X-Axis: Horizontal axis.

Y-Axis: Vertical axis.

You use glVertex2i(x, y) to specify the vertices of your polygons and lines in this 2D space.

**The primitives used to render**

The primitives we used consisted of polygons and lines.

GL\_POLYGON: This primitive is used to draw polygons, which are flat, closed shapes bounded by three or more line segments. In your code, GL\_POLYGON is used to draw various parts of the window, the wall, the floor, the table, and the TV. Each glVertex2i(x, y) call specifies a vertex of the polygon, and the vertices are connected in the order they are called.

GL\_LINES: This primitive is used to draw a series of unconnected line segments. In your code, GL\_LINES is used to draw the window blinds. Each pair of glVertex2i(x, y) calls specifies a line segment.

Some challenges we faced were creating the complex geometric primitives specifically for the switch controllers. This is due to the more natural shape and angle of the controllers.

**Shaders Used**

We used mapping to manually insert the shaders into the picture specifically in the table halves and the wall specifically the corner of the wall. To do this we used fragment shader which is the process where each pixel handles color calculations, lighting, and texture mapping. A potential future approach to improve the shading would use a shader-based approach. This shader provides flexibility and control over the graphics pipeline, enabling advanced effects like programmable lighting, shadows, and texture mapping. They are essential for 3D graphics and are a standard in modern OpenGL development.