Project 4: Render Your Scene With Primitives

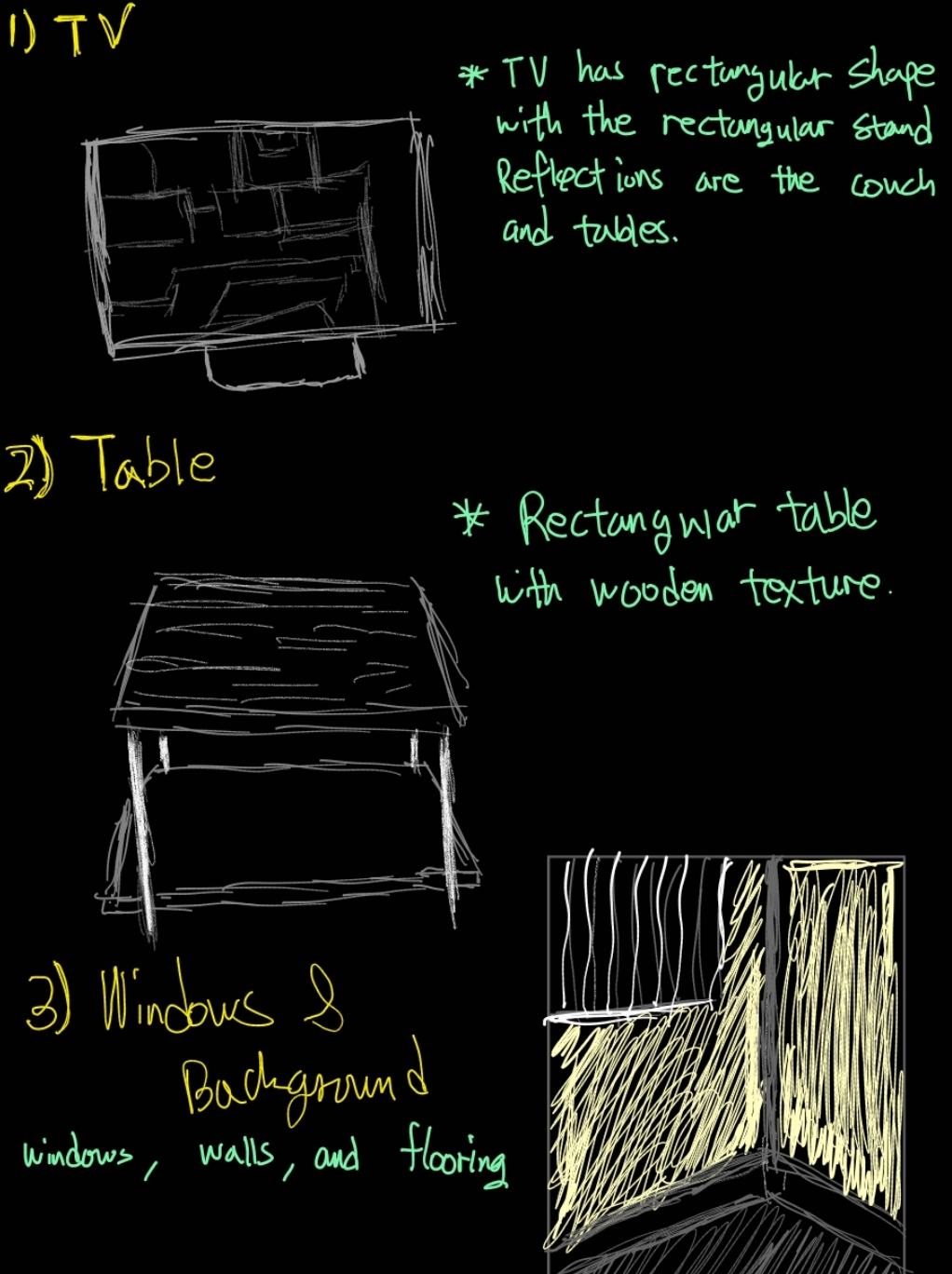
Caleb Klinger, Kyungchan Im (Chris)

Grand Canyon University

CST - 435 : Computer Graphics Lecture & Lab

Ricardo Citro

October 8, 2023

Hand-drawn scene created in Project 3 and reproduce each object using primitives. Our hand drawn scene does not contain the outside of the windows as that will be an applied texture in the future rather than a group of polygons and lines.  


Original compared to Drawing Compared to OpenGL



**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.

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.

**Mathematical characteristics:**

Some mathematical characteristics of the openGL program include coordinate system, color encoding, geometric primitives, color encoding, transformation, projection, and   
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.

Color Encoding:

Colors are encoded using RGBA (Red, Green, Blue, Alpha) and are specified using glColor4b(r, g, b, a). The mathematical characteristic here is the normalization of color values. Typically, colors are represented in the range [0, 255], but OpenGL expects them in a normalized form (i.e., in the range [0, 1]). Your code uses a peculiar scaling factor (127/255), which might be intended to achieve a specific visual effect.

Geometric Primitives

Polygons are defined by a set of vertices. The vertices are connected in the order they are defined, forming a closed shape. Mathematically, a polygon with vertices (x1,y1),(x2,y2),…,(xn,yn)(x1​,y1​),(x2​,y2​),…,(xn​,yn​) in 2D space defines a piecewise linear curve. While lines are defined by two points. A line from (x1,y1)(x1​,y1​) to (x2,y2)(x2​,y2​) can be represented by the line equation in a 2D space.

**Transformations Used:**

Transformations are translations, rotations and scaling which is a necessary concept for operations in graphics rendering. Projection transformations are also a mathematical characteristic used in our code specifically in gluOrtho2D(left, right, bottom, top) to define an orthographic projection. It creates a viewing volume that is a box, and anything inside this box is rendered onto the screen. An example from the code is this orthographic projection gluOrtho2D(0.0, 377, 0.0, 463); which means it defines a box-like viewing volume in the 2D space. Anything inside this box will be rendered onto the screen. The parameters (0.0, 377, 0.0, 463) define the left, right, bottom, and top of this box, respectively. Essentially, it maps the coordinate system defined by these parameters to the window's viewport. glViewport(0, 0, 377, 463); glViewport() sets up how the normalized coordinates are mapped to window coordinates.

The parameters (0, 0, 377, 463) define the lower-left corner of the viewport and the width and height of the viewport in pixels, respectively. This means that the output will be mapped to the entire window. An implicit transformation while explicit transformations are not utilized, the vertex coordinates specified in glVertex2i(x, y) implicitly determine the position and size of the objects. For example:

glVertex2i(112, 463 - 303);

glVertex2i(312, 463 - 287);

These vertex coordinates implicitly define the position and shape of the polygon.

Some other less notable transformations used are

Translation: Moving objects from one place to another.

Rotation: Rotating objects around an axis.

Scaling: Changing the size of objects.

These can be achieved using transformation matrices in OpenGL.

glTranslatef(x, y, z): Translates objects along the x, y, and z axes.

glRotatef(angle, x, y, z): Rotates objects around the x, y, or z axis by a specified angle.

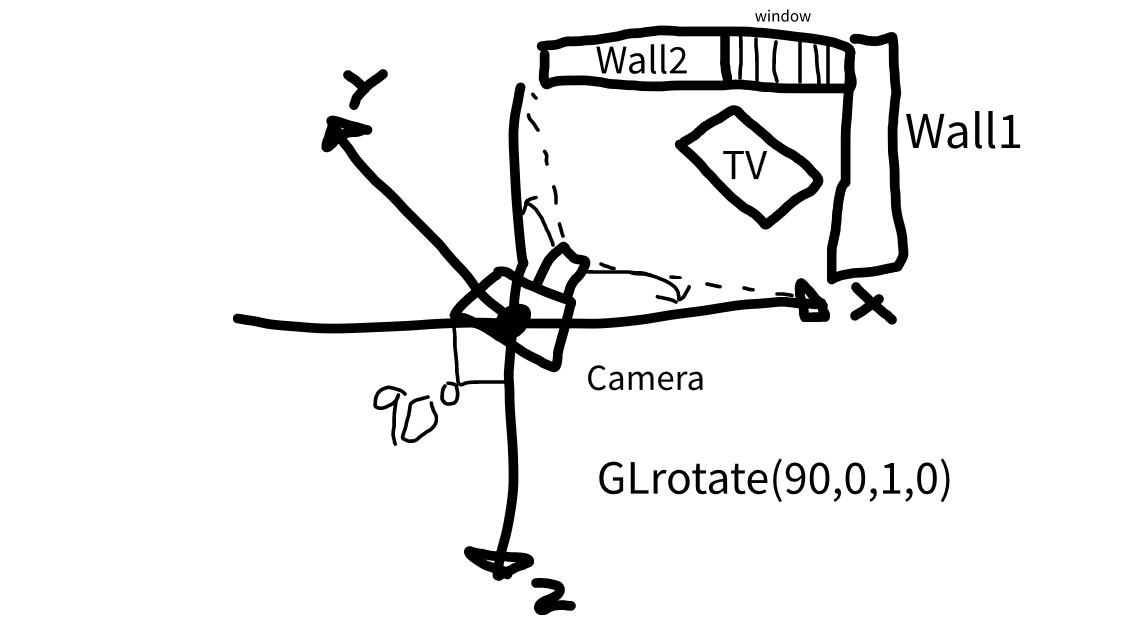
glScalef(x, y, z): Scales objects in the x, y, and z directions.

**Shaders:**

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.

**Camera:**

This is our plan for the camera position and movement of the scene. It will be a rotation of 90 deg from the corner of each wall.



Github: