CAPITULO 2

OBJETOS GEOMÉTRICOS Y TRANSFORMACIONES

2.3 Transformaciones Geométricas en 3D
2.4 Visualización en 3D
2.5 Proyecciones, eliminación de superficies ocultas

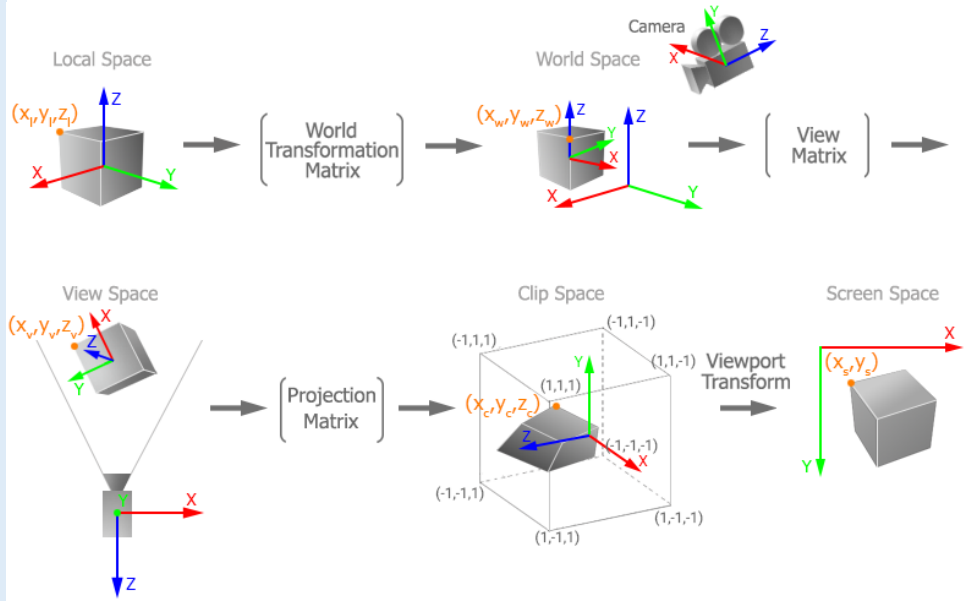
T3D – Going 3D

Exercise 11 Task 1:

Test the 3D model using the projection, view and model matrix in OpenGL

- Use an updated Shader class (learnopengl/shader.h) that includes sending Matrix to shader.

```
//create transformations
glm::mat4 model       = glm::mat4(1.0f); // make sure to initialize matrix to identity matrix first
glm::mat4 view        = glm::mat4(1.0f);
glm::mat4 projection  = glm::mat4(1.0f);

model = glm::rotate(model, glm::radians(-55.0f), glm::vec3(1.0f, 0.0f, 0.0f));
view = glm::translate(view, glm::vec3(0.0f, 0.0f, -3.0f));
projection = glm::perspective(glm::radians(45.0f), (float)SCR_WIDTH / (float)SCR_HEIGHT, 0.1f, 100.0f);

//retrieve teh matrix uniform locations
unsigned int modelLoc = glGetUniformLocation(ourShader.ID, "model");
unsigned int viewLoc = glGetUniformLocation(ourShader.ID, "view");

// pass them to the shaders (3 different ways)
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
 glUniformMatrix4fv(viewLoc, 1, GL_FALSE, &view[0][0]);
// note: currently we set the projection matrix each frame, but since the projection matrix rarely
//changes it's often best practice to set it outside the main loop only once.
ourShader.setMat4("projection", projection);
```

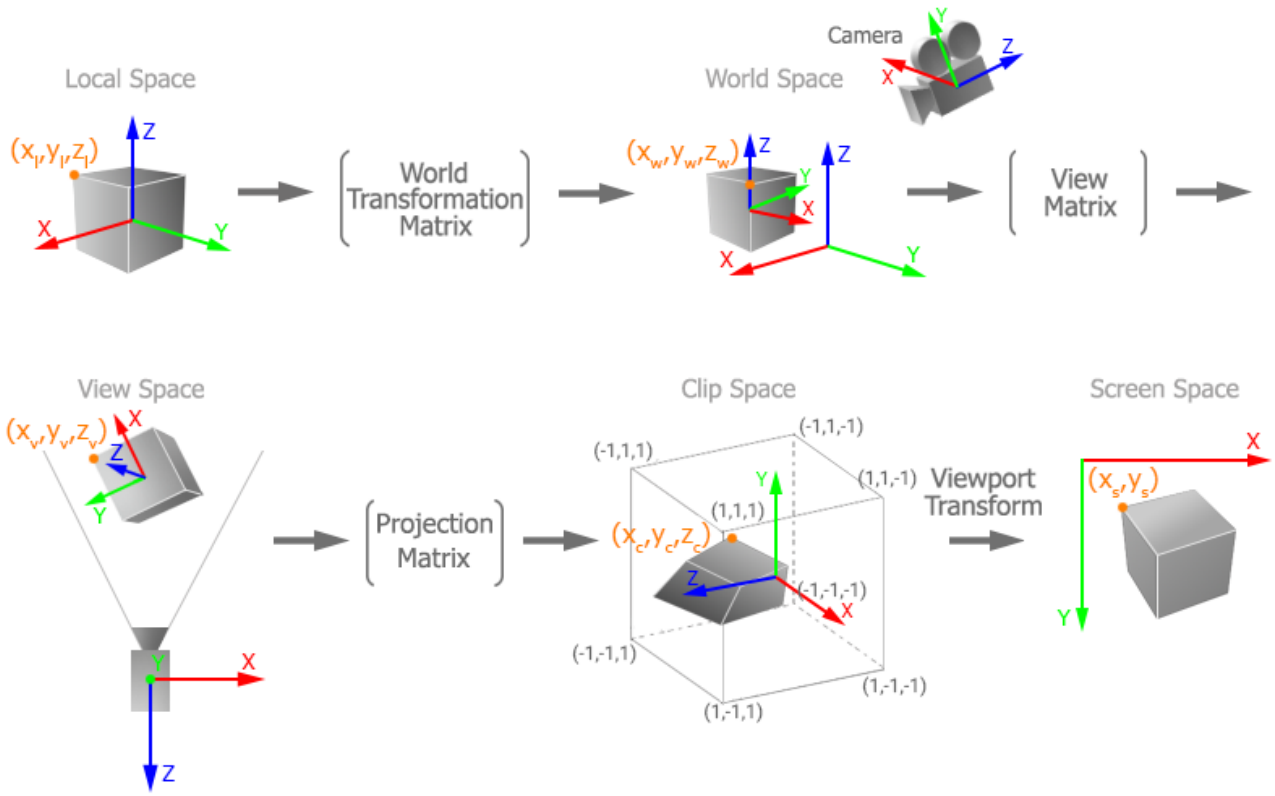


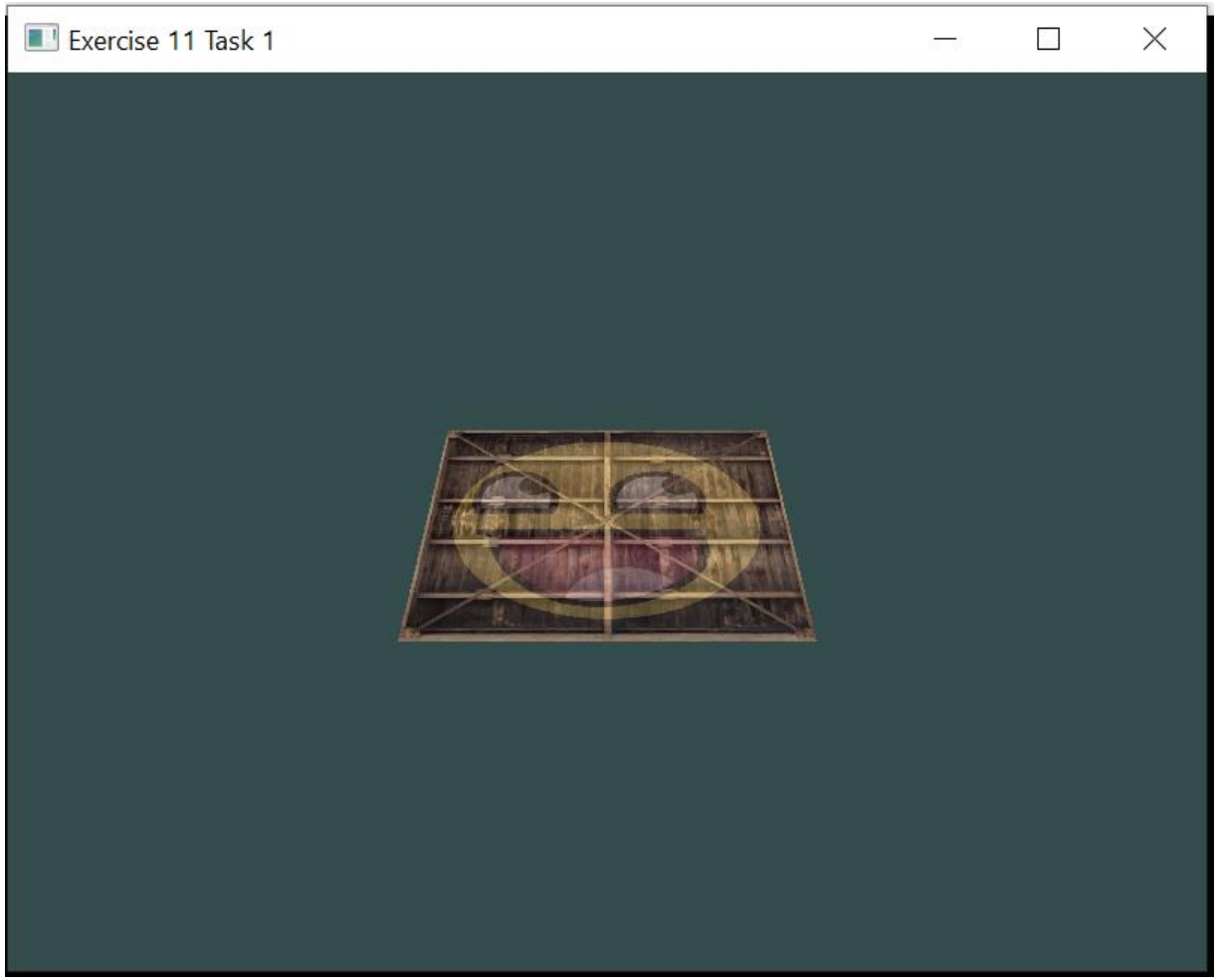
T3D – Going 3D

Exercise 11 Task 1:

Test the 3D model using the projection, view and model matrix in OpenGL

- Use an updated Shader class that includes sending Matrix to shader.



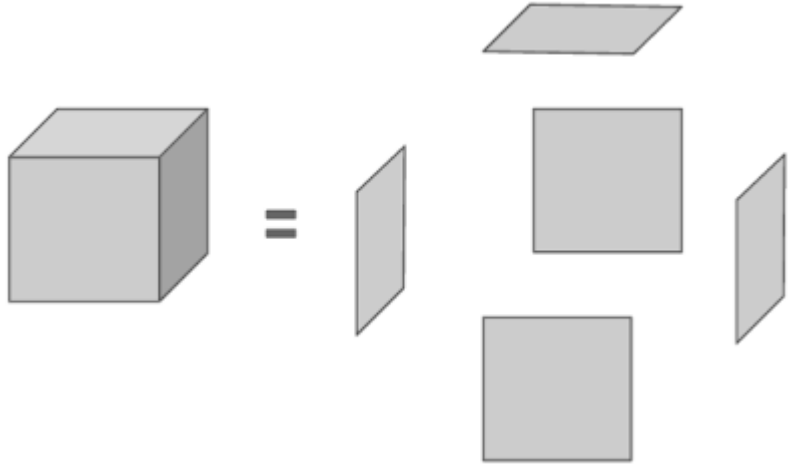


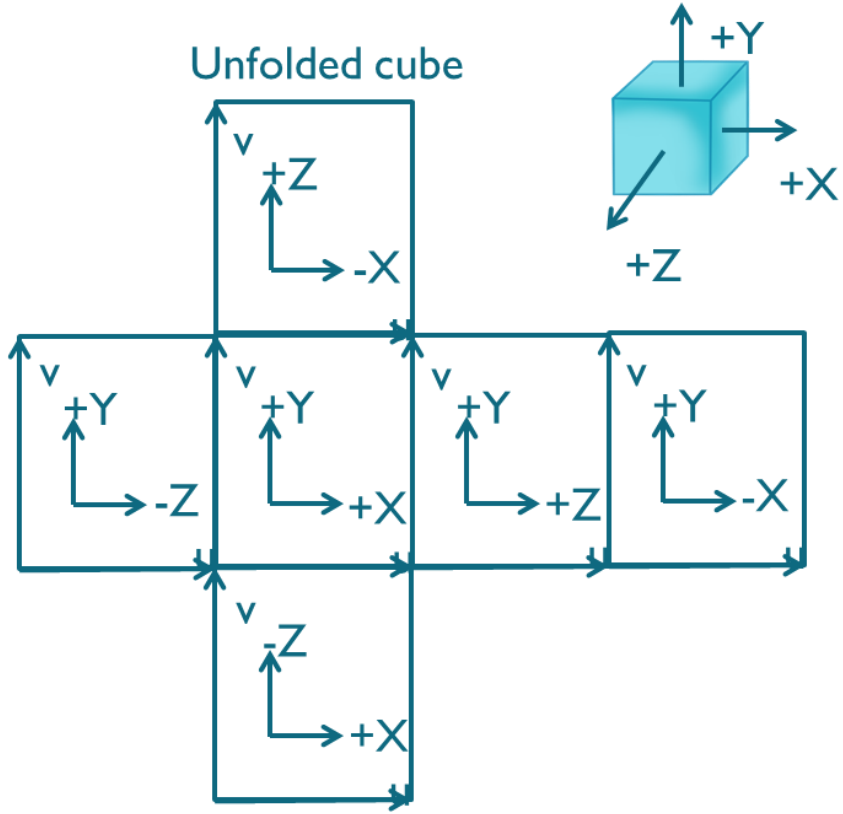
T3D – Going 3D

Exercise 11 Task 2: Test the 3D model of a 3D object (CUBE) and rotate it over the time.

- Set the vertices (source: cube_vertices.txt)

```
float vertices[] = {
    -0.5f, -0.5f, -0.5f,  0.0f, 0.0f,
     0.5f, -0.5f, -0.5f,  1.0f, 0.0f,
     0.5f,  0.5f, -0.5f,  1.0f, 1.0f,
     0.5f,  0.5f, -0.5f,  1.0f, 1.0f,
    -0.5f,  0.5f, -0.5f,  0.0f, 1.0f,
    -0.5f, -0.5f, -0.5f,  0.0f, 0.0f,

    -0.5f, -0.5f,  0.5f,  0.0f, 0.0f,
     0.5f, -0.5f,  0.5f,  1.0f, 0.0f,
     0.5f,  0.5f,  0.5f,  1.0f, 1.0f,
     0.5f,  0.5f,  0.5f,  1.0f, 1.0f,
    -0.5f,  0.5f,  0.5f,  0.0f, 1.0f,
    -0.5f, -0.5f,  0.5f,  0.0f, 0.0f,

    -0.5f,  0.5f,  0.5f,  1.0f, 0.0f,
    -0.5f,  0.5f, -0.5f,  1.0f, 1.0f,
    -0.5f, -0.5f, -0.5f,  0.0f, 1.0f,
    -0.5f, -0.5f, -0.5f,  0.0f, 1.0f,
    -0.5f, -0.5f,  0.5f,  0.0f, 0.0f,
    -0.5f,  0.5f,  0.5f,  1.0f, 0.0f,

     0.5f,  0.5f,  0.5f,  1.0f, 0.0f,
     0.5f,  0.5f, -0.5f,  1.0f, 1.0f,
     0.5f, -0.5f, -0.5f,  0.0f, 1.0f,
     0.5f, -0.5f, -0.5f,  0.0f, 1.0f,
     0.5f, -0.5f,  0.5f,  0.0f, 0.0f,
     0.5f,  0.5f,  0.5f,  1.0f, 0.0f,

    -0.5f, -0.5f, -0.5f,  0.0f, 1.0f,
     0.5f, -0.5f, -0.5f,  1.0f, 1.0f,
     0.5f, -0.5f,  0.5f,  1.0f, 0.0f,
     0.5f, -0.5f,  0.5f,  1.0f, 0.0f,
    -0.5f, -0.5f,  0.5f,  0.0f, 0.0f,
    -0.5f, -0.5f, -0.5f,  0.0f, 1.0f,

    -0.5f,  0.5f, -0.5f,  0.0f, 1.0f,
     0.5f,  0.5f, -0.5f,  1.0f, 1.0f,
     0.5f,  0.5f,  0.5f,  1.0f, 0.0f,
     0.5f,  0.5f,  0.5f,  1.0f, 0.0f,
    -0.5f,  0.5f,  0.5f,  0.0f, 0.0f,
    -0.5f,  0.5f, -0.5f,  0.0f, 1.0f
};
```





T3D – Going 3D

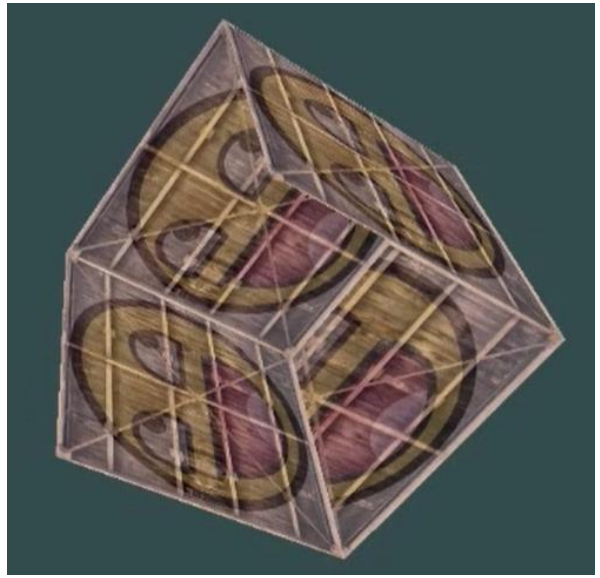
Exercise 11 Task 2: Test the 3D model of a 3D object (CUBE) and rotate it over the time.

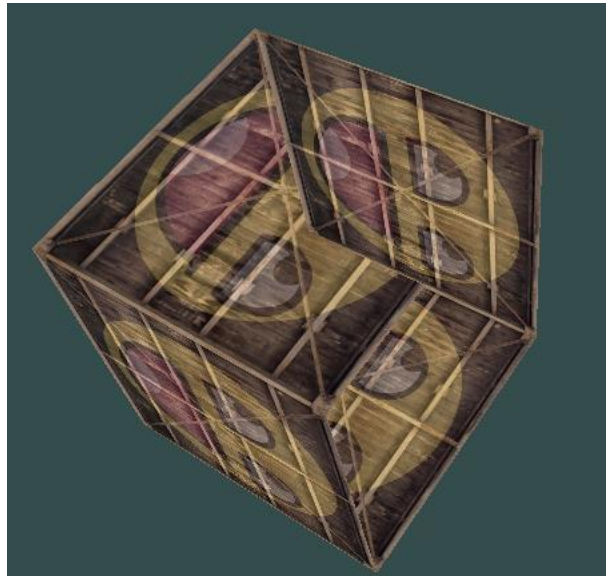
- Rotate over the time

```
model = glm::rotate(model, (float)glfwGetTime() * glm::radians(50.0f), glm::vec3(0.5f, 1.0f, 0.0f));
```

- Draw the cube using glDrawArrays (as we didn't specify indices), but this time with a count of 36 vertices.

```
glDrawArrays(GL_TRIANGLES, 0, 36);
```

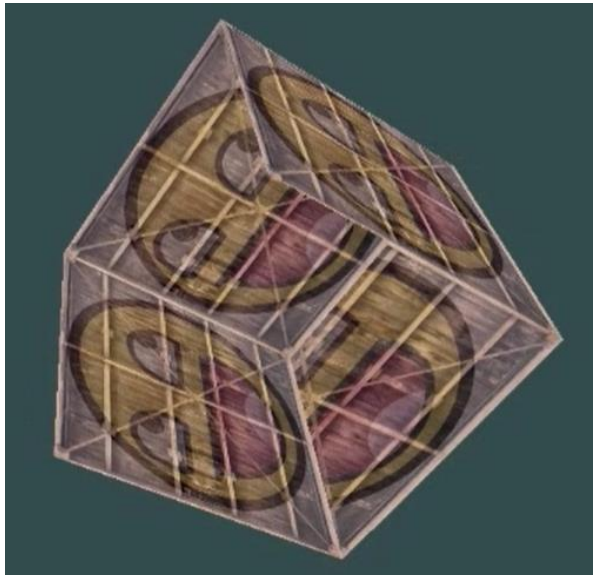


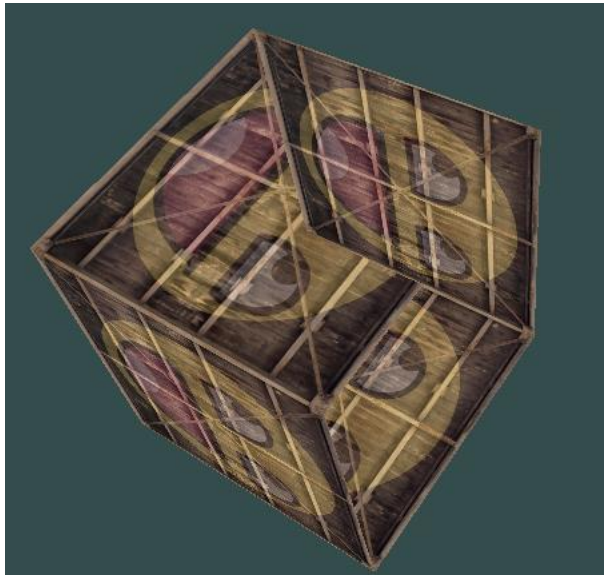


T3D – Going 3D

Some sides of the cubes are being drawn over other sides of the cube.

This happens because when OpenGL draws your cube triangle-by-triangle, fragment by fragment, it will **overwrite any pixel color that might've already been drawn there before.**





Since OpenGL gives no guarantee on the order of triangles rendered (within the same draw call), some triangles are drawn on top of each other even though one should clearly be in front of the other.

Luckily, OpenGL stores depth information in a buffer called the **z-buffer** that allows OpenGL to decide **when to draw over a pixel and when not to**. Using the z-buffer we can configure OpenGL to do depth-testing.

T3D – Going 3D – Z buffer

- OpenGL stores all its depth information in a **z-buffer**, also known as a **depth buffer**.
- GLFW automatically creates such a buffer for you (just like it has a color-buffer that stores the colors of the output image).
- The depth is stored within each fragment (as the **fragment's z value**) and whenever the fragment wants to output its color, **OpenGL compares its depth values with the z-buffer**.

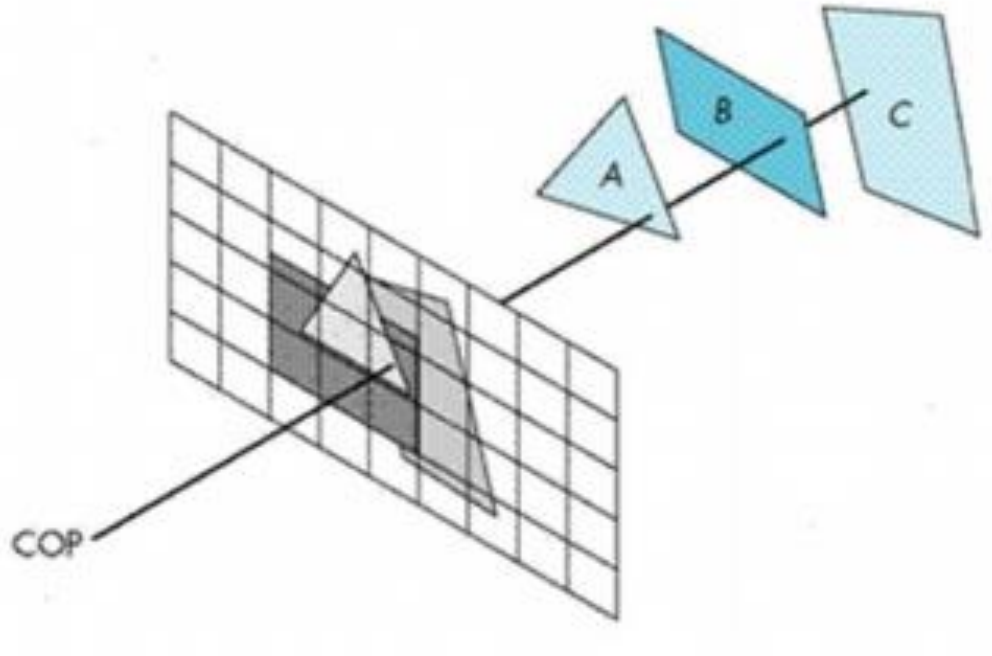


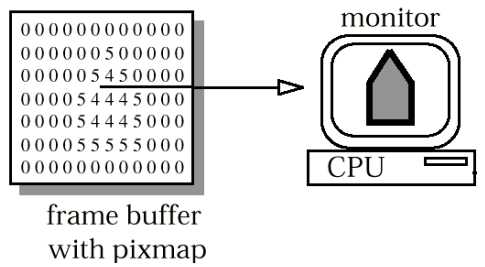
Fig.3.3. Z- buffer algorithm

If the current fragment is behind the other fragment it is discarded, otherwise overwritten. This process is called depth testing and is done automatically by OpenGL.

Z-Buffer Algorithm

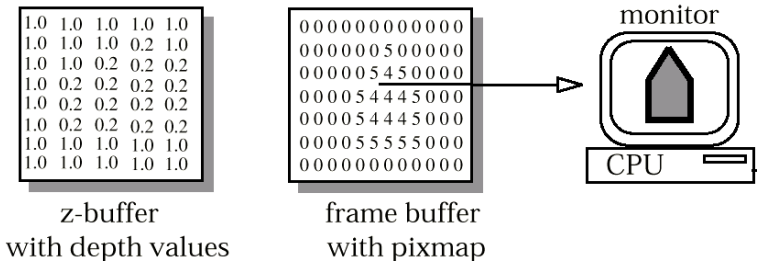
About

- Recall, frame/refresh buffer:
  - Screen is refreshed one scan line at a time, from pixel information held in a refresh or *frame buffer*



- Additional buffers can be used to store other pixel information
  - E.g., double buffering for animation
    - 2nd frame buffer to which to draw an image (which takes a while)
    - then, when drawn, switch to this 2nd frame/refresh buffer and start drawing again in 1st

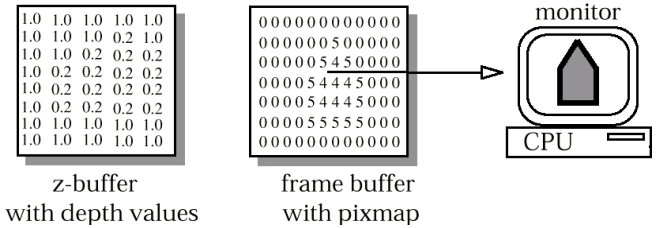
- Also, a z-buffer in which z-values (depth of points on a polygon) stored for VSD



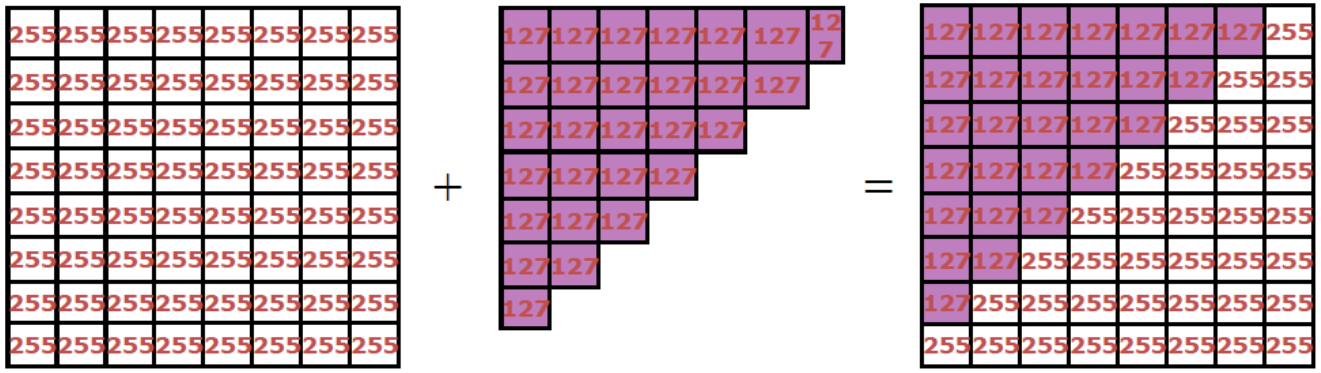
Z-Buffer Algorithm

Overview

- Just draw every polygon
  - If find a piece (one or more pixels) of a polygon is closer to the front of what there already, draw over it



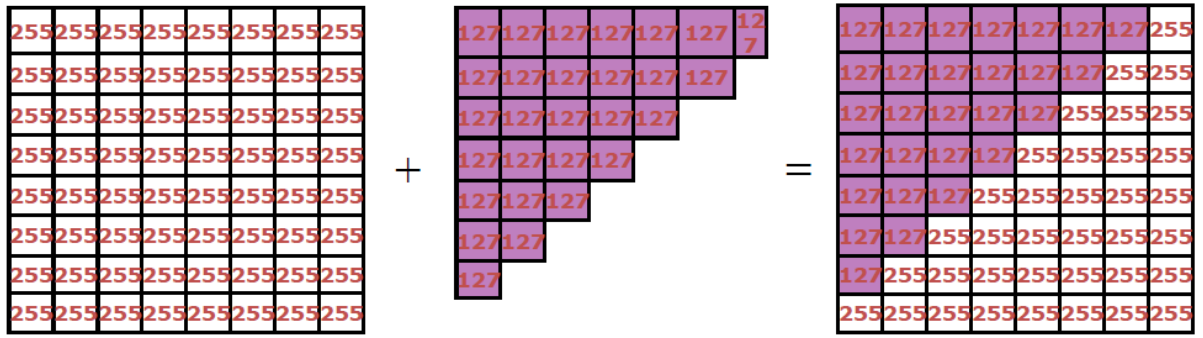
- Init Z-buffer to background value
  - furthest plane view vol., e.g, 255, 8-bit
- Polygons scan-converted in arbitrary order
  - When pixels overlap, use Z-buffer to decide which polygon "gets" that pixel
    - If new point has z values less than previous one (i.e., closer to the eye), its z-value is placed in the z-buffer and its color placed in the frame buffer at the same (x,y)
    - Otherwise the previous z-value and frame buffer color are unchanged
  - Below shows numeric z-values and color to represent fb values



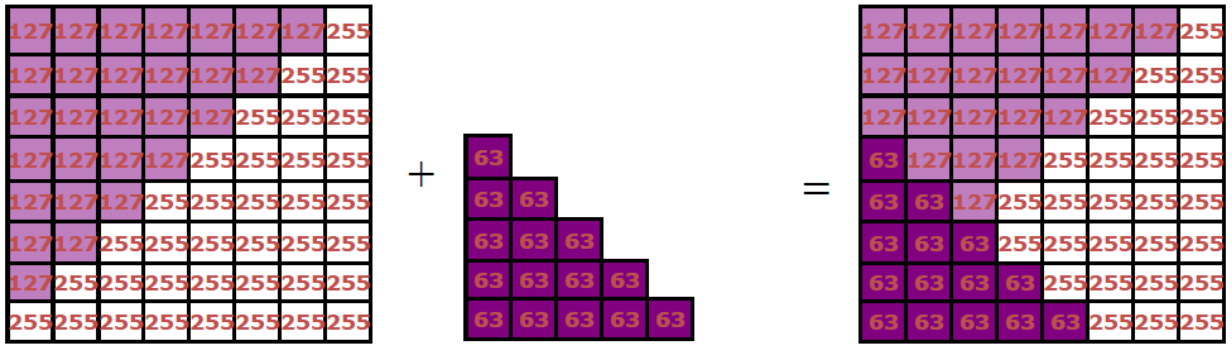
Z-Buffer Algorithm

Example

- Polygons scan-converted in arbitrary order
- After 1st polygon scan-converted, at depth 127



- After 2nd polygon, at depth 63 – in front of some of 1st polygon



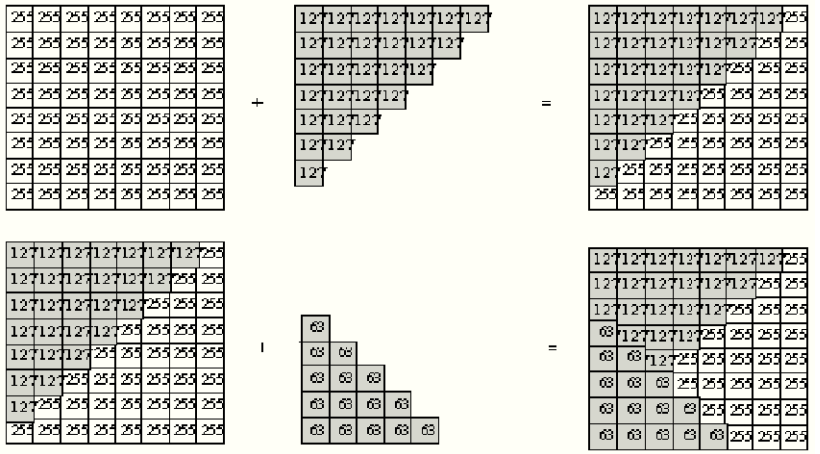
Z-Buffer Algorithm

Pseudocode

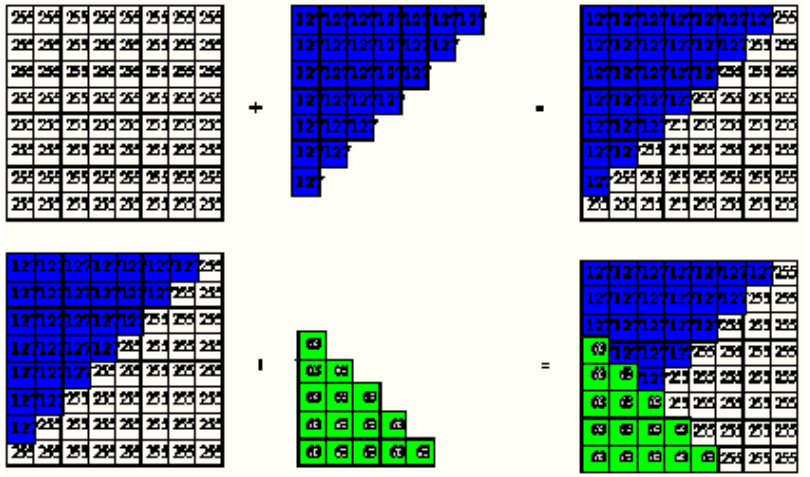
- Algorithm again:
  - Draw every polygon that we can't reject trivially
  - "If find piece of polygon closer to front, paint over whatever was behind it"

```
void zBuffer()
    {
    // Initialize to "far"
    for ( y = 0; y < YMAX; y++)
        for ( x = 0; x < XMAX; x++) {
            WritePixel (x, y, BACKGROUND_VALUE);
            WriteZ (x, y, 0);
            }
    // Go through polygons
    for each polygon
        for each pixel in polygon's projection {
        // pz = polygon's Z-value at pixel (x, y);
            if ( pz < ReadZ (x, y) ) {
                // New point is closer to front of view
                WritePixel (x, y, poly's color at pixel (x, y));
                WriteZ (x, y, pz);
                }
       }
    }
```

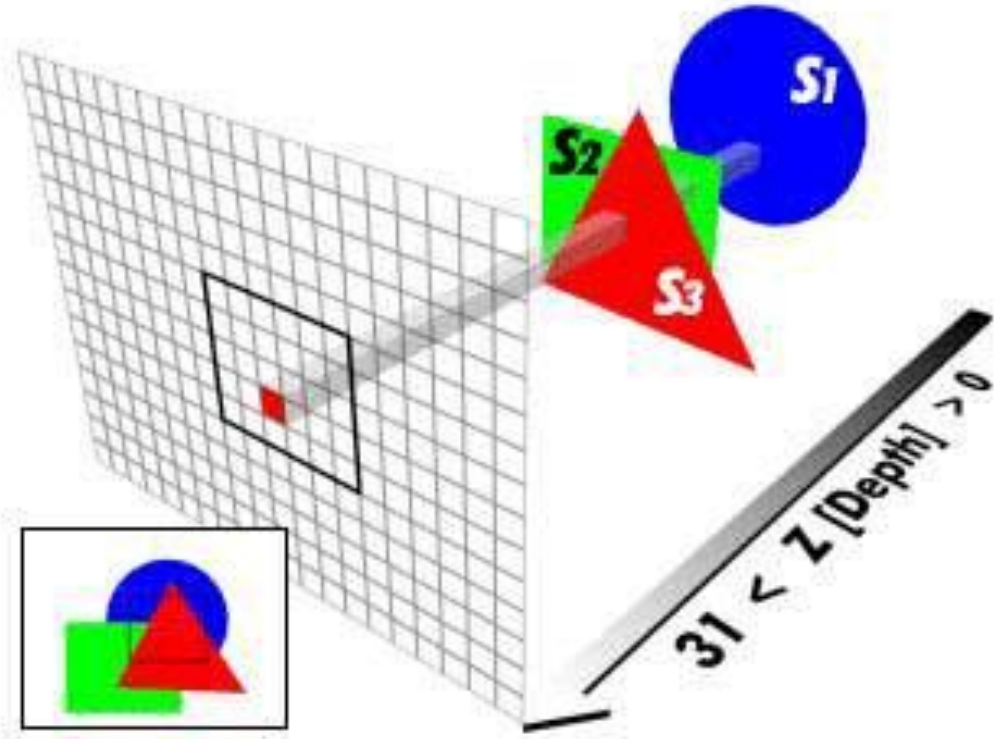
Z buffer holds z values of polygons:



Frame buffer holds values of polygons' colors:



T3D – Going 3D – Z buffer



1

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

2

0	0	0	0	0	0
0	0	0	0	0	0
10	10	10	10	0	0
10	10	10	10	0	0
10	10	10	10	0	0

3

5	5	5	5	5	5
5	5	5	5	5	5
10	10	10	10	5	5
10	10	10	10	5	5
10	10	10	10	5	5

4

5	5	15	15	5	5
5	5	15	15	15	5
10	15	15	15	15	15
10	15	15	15	15	15
15	15	15	15	15	15

T3D – Going 3D – Z buffer

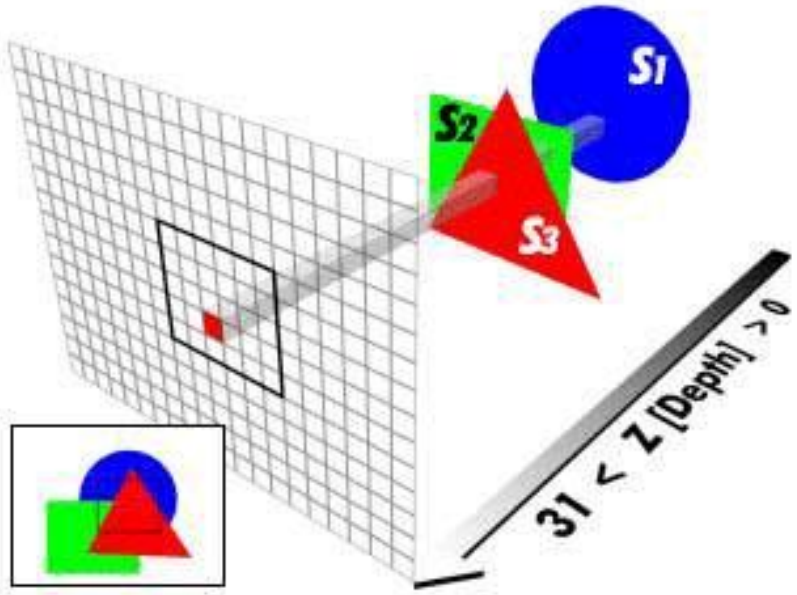
We can enable depth testing using glEnable. The glEnable and glDisable functions allow us to enable/disable certain functionality in OpenGL.

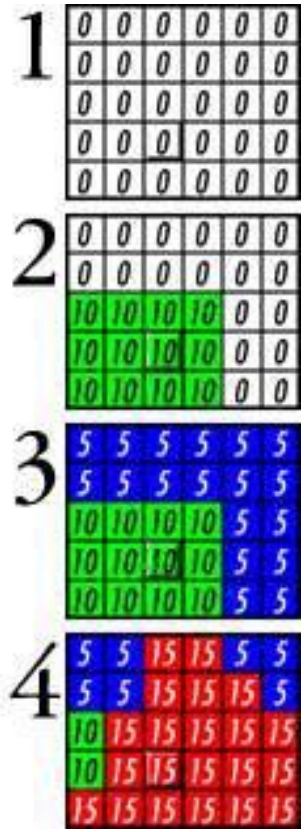
- That functionality is then enabled/disabled until another call is made to disable/enable it.
- Right now we want to enable depth testing by enabling GL_DEPTH_TEST:

```
glEnable(GL_DEPTH_TEST);
```

- Since we're using a depth buffer we also want to clear the depth buffer before each render iteration (otherwise the depth information of the previous frame stays in the buffer).
- Just like clearing the color buffer, we can clear the depth buffer by specifying the DEPTH_BUFFER_BIT bit in the glClear function:

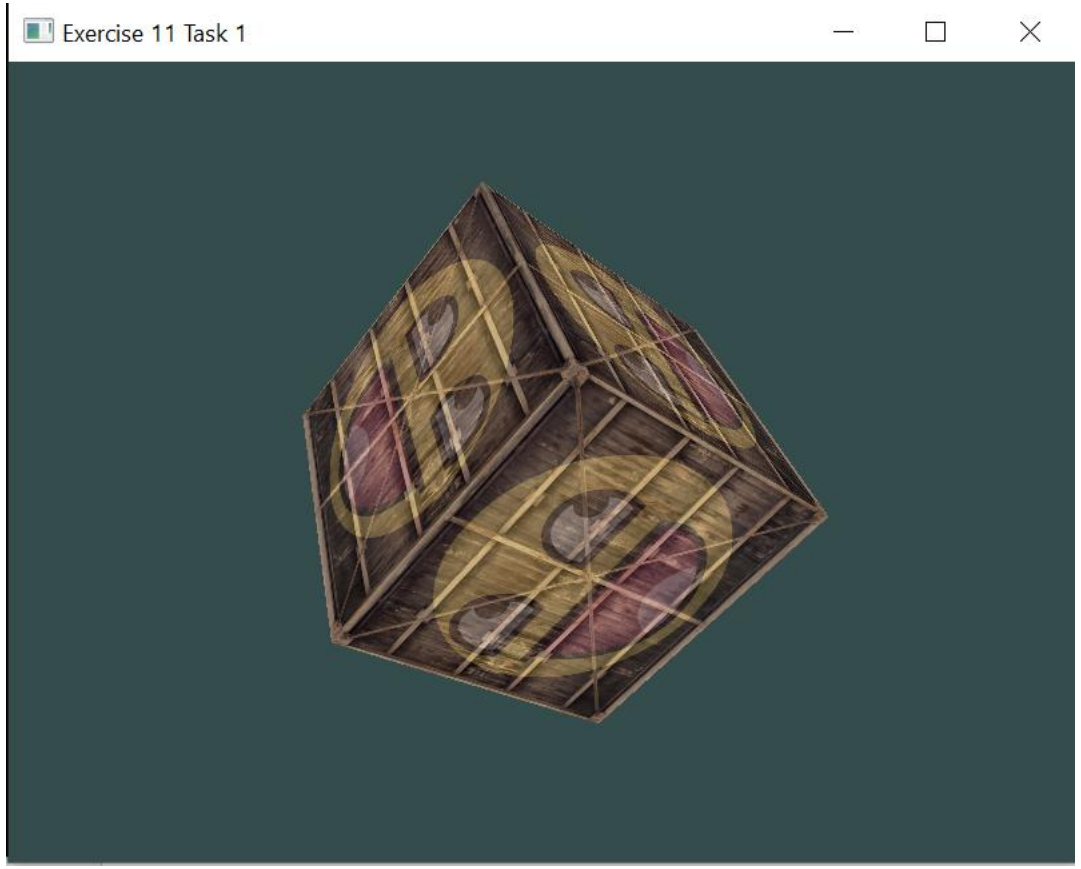
```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

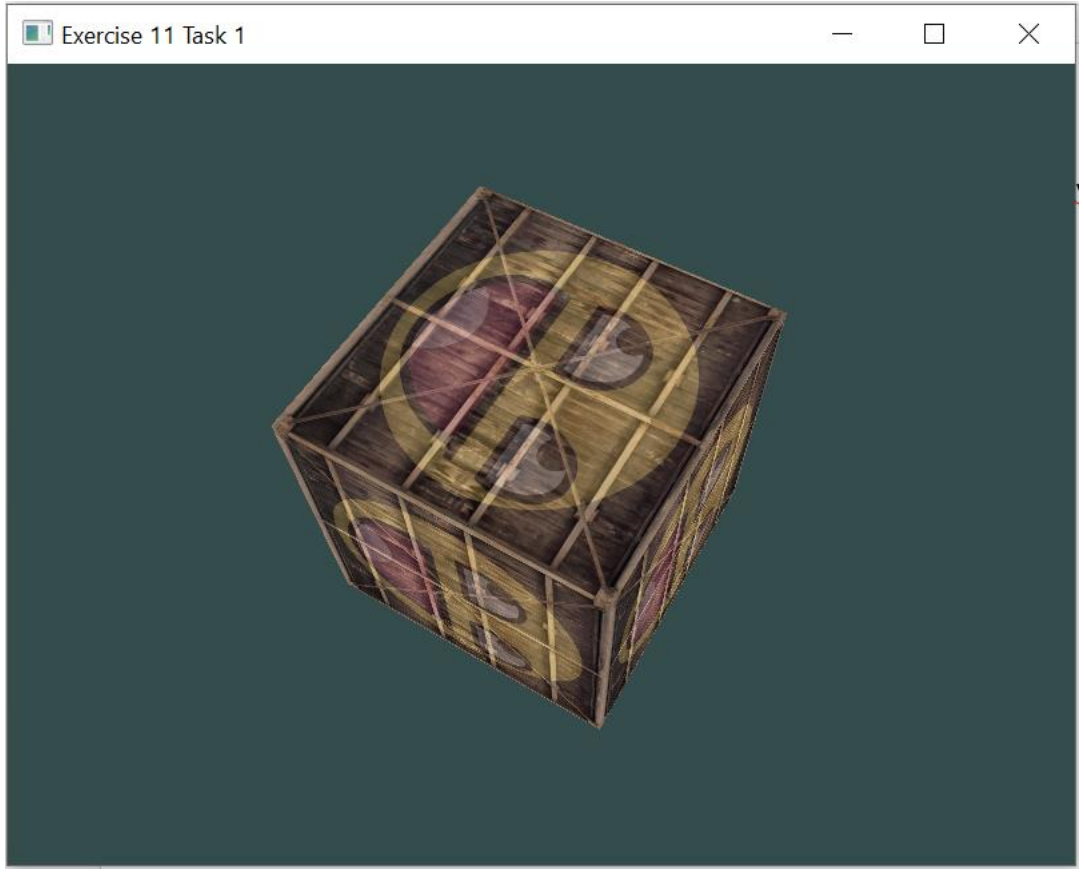




T3D – Going 3D – Z buffer

Exercise 11 Task 3: Test the 3D model of a 3D object (CUBE) and rotate it over the time using the **Z buffer Option**.





T3D – Going 3D

Exercise 11 Task 4: Display 10 of our cubes on screen.

- Each cube will look the **same** but will only differ in where **it's located in the world with each a different rotation**.
- The graphical layout of the cube is already defined so **we don't have to change our buffers or attribute arrays** when rendering more objects.
- The only thing we have to **change** for each object is its **model matrix** where we transform the cubes into the world.
- First, let's define a translation vector for each cube that specifies its position in world space. We'll define 10 cube positions in a glm::vec3 array:

```
glm::vec3 cubePositions[] = {
    glm::vec3( 0.0f,  0.0f,  0.0f),
    glm::vec3( 2.0f,  5.0f, -15.0f),
    glm::vec3(-1.5f, -2.2f, -2.5f),
    glm::vec3(-3.8f, -2.0f, -12.3f),
    glm::vec3( 2.4f, -0.4f, -3.5f),
    glm::vec3(-1.7f,  3.0f, -7.5f),
    glm::vec3( 1.3f, -2.0f, -2.5f),
    glm::vec3( 1.5f,  2.0f, -2.5f),
    glm::vec3( 1.5f,  0.2f, -1.5f),
    glm::vec3(-1.3f,  1.0f, -1.5f)
};
```

T3D – Going 3D

Exercise 11 Task 4: Display 10 of our cubes on screen.

- Now, within the render loop we want to call glDrawArrays 10 times, but this time send a different model matrix to the vertex shader each time before we send out the draw call.

- We will create a small loop within the render loop that renders our object 10 times with a different model matrix each time. Note that we also add a **small unique rotation** to each container.

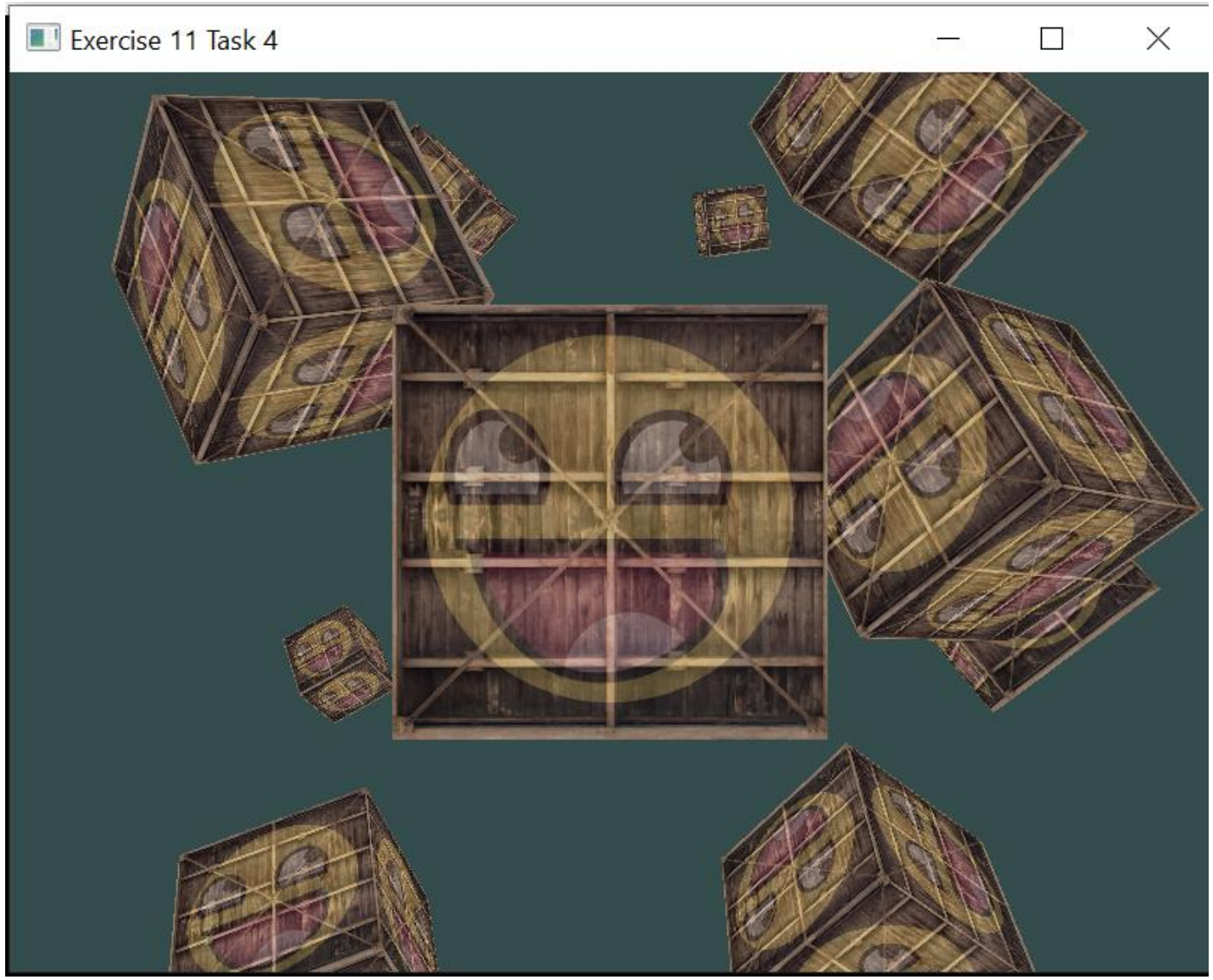
```
glBindVertexArray(VAO);
for(unsigned int i = 0; i < 10; i++)
{
    glm::mat4 model = glm::mat4(1.0f);
    model = glm::translate(model, cubePositions[i]);
    float angle = 20.0f * i;
    model = glm::rotate(model, glm::radians(angle), glm::vec3(1.0f, 0.3f, 0.5f));
    ourShader.setMat4("model", model);

    glDrawArrays(GL_TRIANGLES, 0, 36);
}
```

T3D – Going 3D

Exercise 11 Task 4: Display 10 of our cubes on screen.

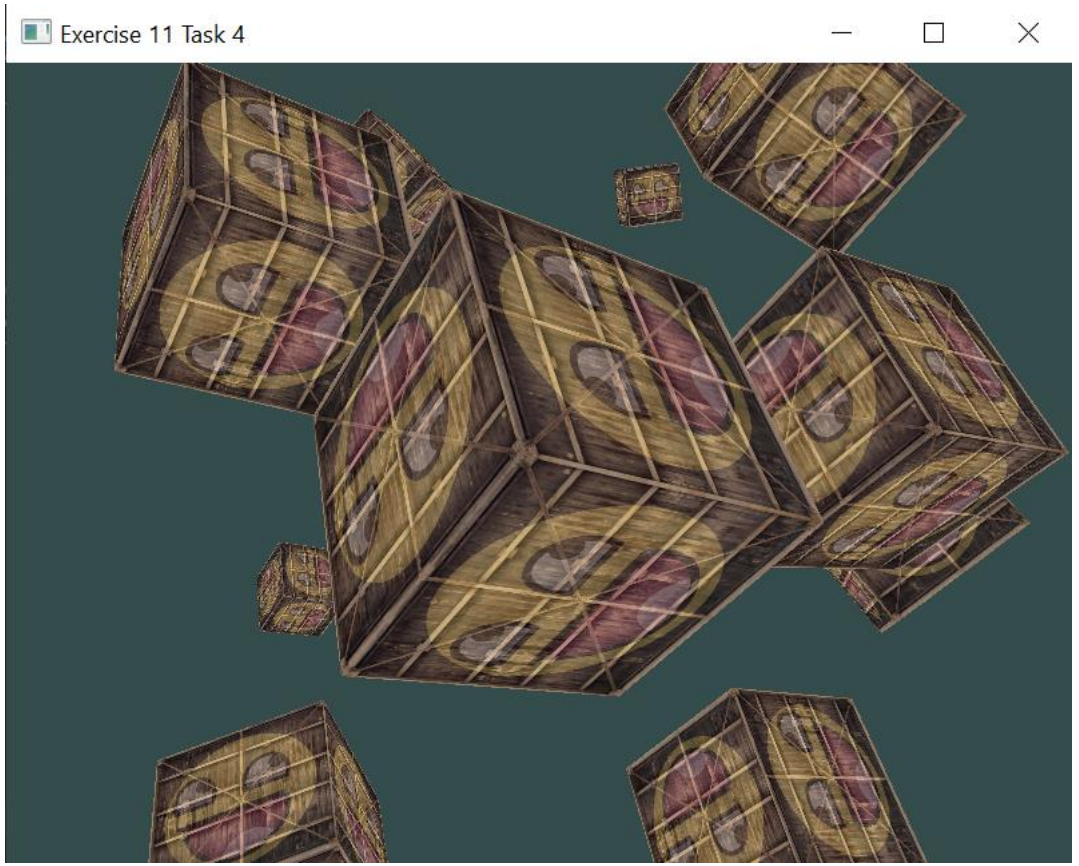
- Try experimenting with the FoV and aspect-ratio parameters of GLM's projection function. See if you can figure out how those affect the perspective frustum.

- Play with the view matrix by translating in several directions and see how the scene changes. Think of the view matrix as a camera object.



T3D – Going 3D

Exercise 11 Task 5: Try to make every 3rd container (including the 1st) rotate over time, while leaving the other containers static using just the model matrix.



T3D – Going 3D

Exercise 11 Task 5: Try to make every 3rd container (including the 1st) rotate over time, while leaving the other containers static using just the model matrix.

```
glBindVertexArray(VAO);
for(unsigned int i = 0; i < 10; i++)
{
    // calculate the model matrix for each object and pass it to shader before drawing
    glm::mat4 model = glm::mat4(1.0f);
    model = glm::translate(model, cubePositions[i]);
    float angle = 20.0f * i;
    if(i % 3 == 0)  // every 3rd iteration (including the first) we set the angle using GLFW's time function.
        angle = glfwGetTime() * 25.0f;
    model = glm::rotate(model, glm::radians(angle), glm::vec3(1.0f, 0.3f, 0.5f));
    ourShader.setMat4("model", model);

    glDrawArrays(GL_TRIANGLES, 0, 36);
}
```

T3D – Going 3D

Exercise 11 Task 5: Try to make every 3rd container (including the 1st) rotate over time, while leaving the other containers static using just the model matrix.