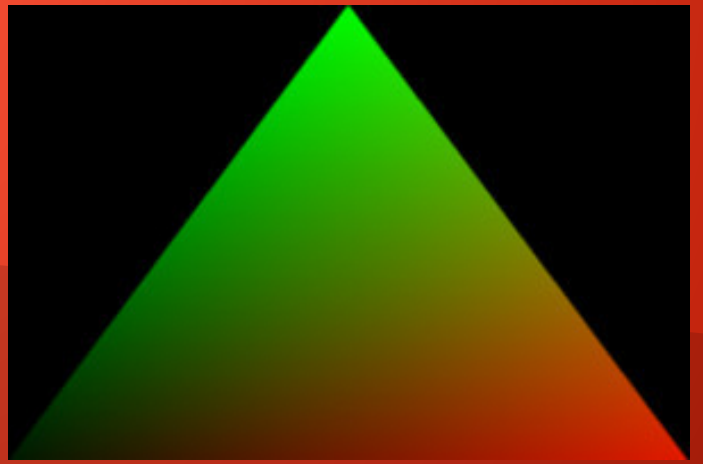
Interpolation, Indexed Draws and Projections

Interpolation

- Happens during the Rasterization stage.
- Per-vertex attributes passed on are “interpolated” using the other values on the primitive.
- In other words: A weighted average of the three verices on a triangle is passed on
- Fragment Shader picks up the interpolated value and uses that.
- The value is effectively an estimate of what the value should be at that position, had we defined it ourself.

Interpolation

- Classic example: Using position co-ordinates as RGB values.
- Top of triangle is green, because in (x, y, z) the y is high.
- Convert to RGB, then G (green) is high.
- Midway between red and green areas, colours blend, but we didn’t define that vertex position!
- The value was interpolated.



Interpolation

- Interpolation is used for quickly and accurately estimating values without defining them.
- Can be used for interpolating Texture Co-ordinates when mapping textures.
- Can be used for interpolating Normal Vectors when handling lighting.
- Especially useful in Phong Shading to create the illusion of smooth/rounded surfaces.

Indexed Draws

- Define vertices to draw a cube.
- Cube will consist of 12 triangles (two for each face).
- 12 x 3 vertices per triangle = 36 vertices.
- But a cube only has 8 vertices!
- Some will be defined multiple times, very messy!

Indexed Draws

- Instead, why not define the 8 vertices of the cube…
- Number them 1 to 8 (or 0 to 7 in C++)…
- And refer to them by their number!

Indexed Draws

- Just bind them to and Element Array Buffer in the VAO

  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
- Sometimes they're called an "Element" instead of an "Index". They mean the same thing.

- Can still be a bit of a pain…
- Solution: 3D modelling software!
- Load in models (we will do this later in the course).

Projections

- Used to convert from “View Space” to “Clip Space”.
- Can be used to give the scene a ‘3D’ look.
- Alternatively can be used to create a 2D style for projects that require it.
- Need to understand co-ordinate systems...

Projections: Co-ordinate Systems

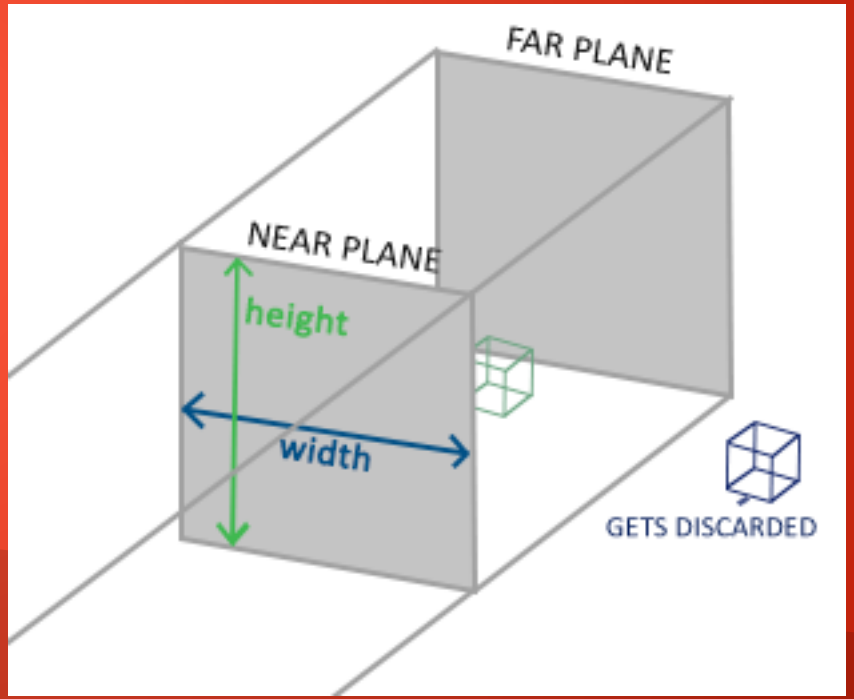
- Local Space: Raw position of each vertex drawn relative to origin. Multiply by Model Matrix to get...
- World Space: Position of vertex in the world itself if camera is assumed to be positioned at the origin. Multiply by View Matrix to get…
- View Space: Position of vertex in the world, relative to the camera position and orientation. Multiply by Projection Matrix to get…
- Clip Space: Position of vertex in the world, relative to the camera position and orientation, as viewed in the area not to be “clipped” from the final output.
- Screen Space: After clipping takes place, the final image is created and placed on the co-ordinate system of the window itself.

Projections

- To create Clip Space we define an area (frustum) of what is not to be clipped with a Projection Matrix.
- Two commonly used types of Projection:
  - Orthographic (most common in 2D applications)
  - Perspective (most common in 3D applications)

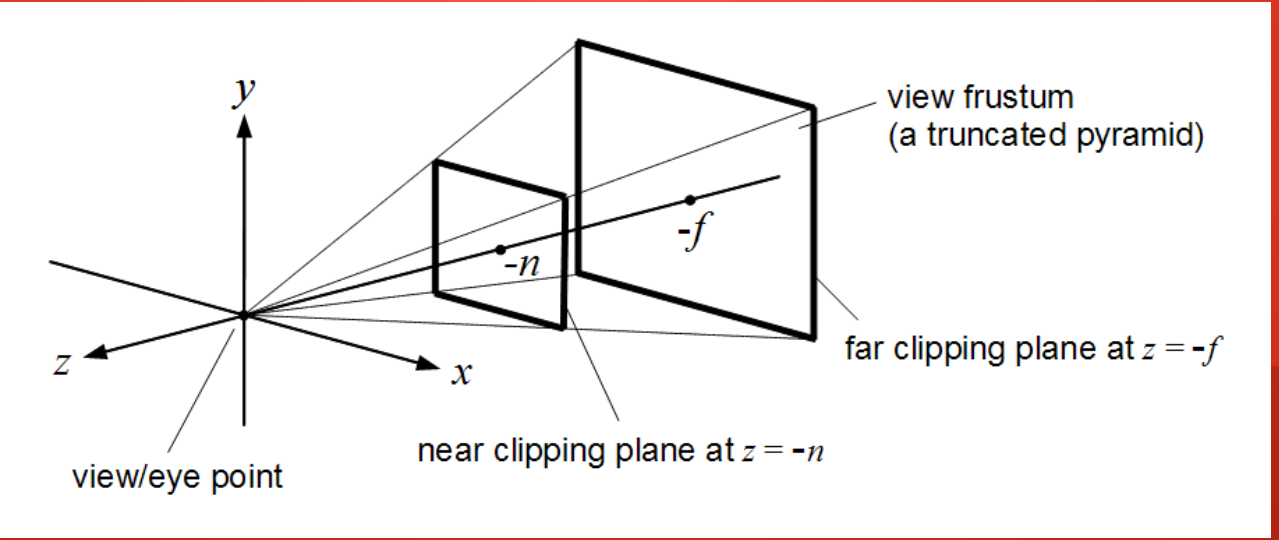
Projections : Orthographic

- The frustrum for orthographic projections is cuboid.
- Everything between the Far Plane and Near Plane is kept. Rest is discarded.
- Parallel nature of Orthographic means 3D depth is not visible.
- Move object closer/further and it won’t change size on screen.

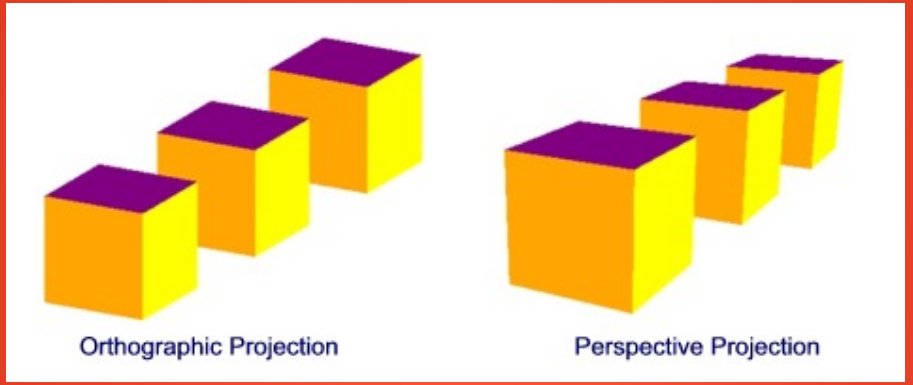


Projections : Perspective

- The frustrum for orthographic projections is a “truncated pyramid”.
- Each pixel on Near Plane diverges at an angle to reach matching point on Far Plane.
- Gives the illusion of depth.



Projections Comparison



- Orthographic: The one furthest back looks to be the same size as the one at the front, implying it’s actually larger.
- Perspective: The one at the back look smaller than the one at the front, due to it being more distant, as it should.

Projections with GLM and OpenGL

- glm::mat4 proj = glm::perspective(fov, aspect, near, far);
- fov = field-of-view, the angle of the frustum.
- aspect = aspect ratio of the viewport (usually its width divided by its height).
- near = distance of the near plane.
- far = distance of the far plane.
- Bind the given matrix to a uniform in the shader.
- gl_Position = projection * view * model * vec4(pos, 1.0);

Summary

- Interpolation calculates weighted values between vertices during rasterization.
- Indexed Draws let us define vertices once then reference them to draw them.
- Projection Matrices convert View Space in to Clip Space.
- Orthographic Projections are used for 2D applications and don’t allow depth perception.
- Perspective Projections are for 3D applications and create the illusion of depth.
- GLM has the glm::perspective function to create perspective matrices.

See you next video!