CSE 167 Fall 2020

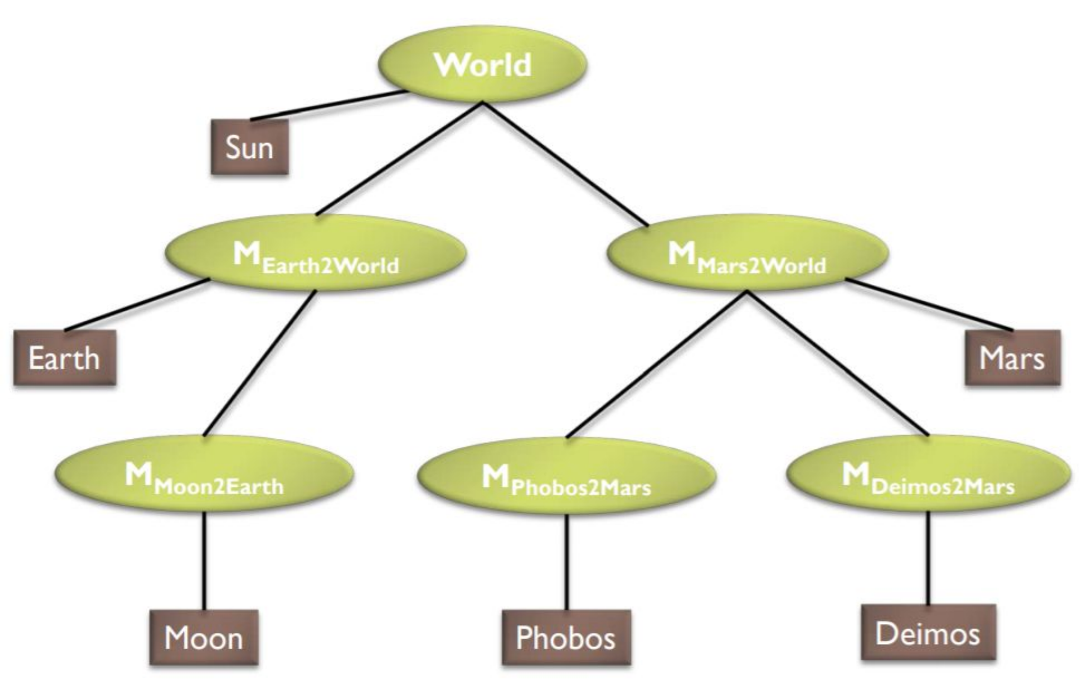
Discussion 6 - Nov. 10, 2020

Project 3

- Project specifications at:
  - http://ivl.calit2.net/wiki/index.php/Project3F20
- Features to implement:
  - texturing
  - sky box
  - environment mapping
  - scene graph

Scene Graph

Scene Graph

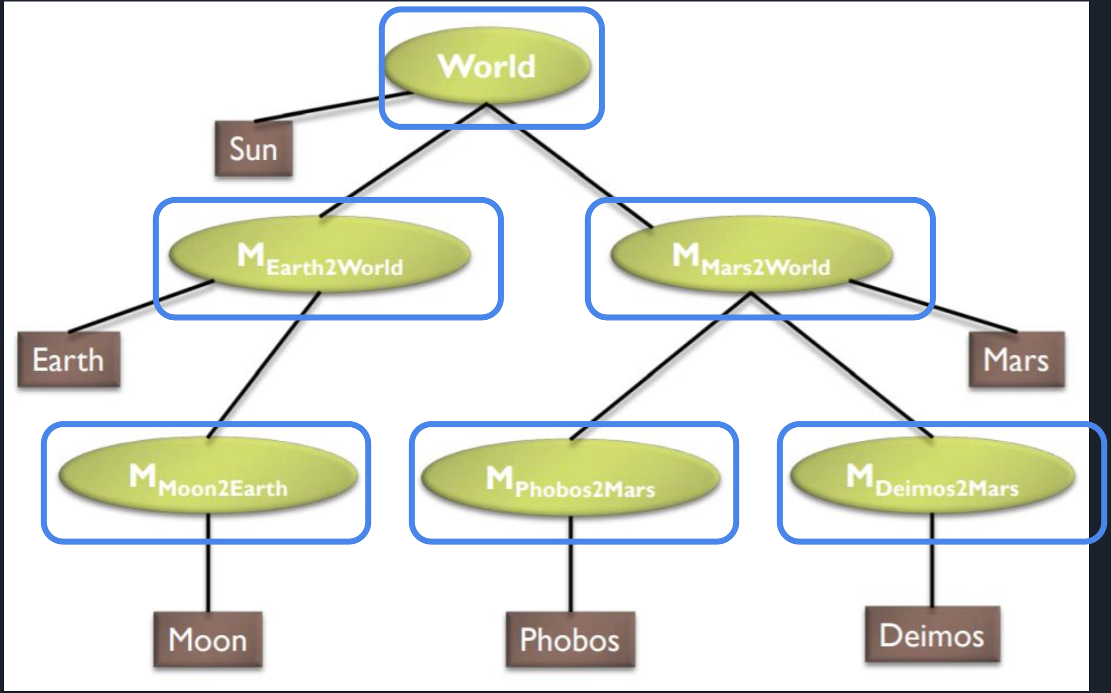

World
Sun
$M_{Earth2World}$
$M_{Mars2World}$
Earth
Mars
$M_{Moon2Earth}$
$M_{Phobos2Mars}$
$M_{Deimos2Mars}$
Moon
Phobos
Deimos

Scene Graph

- Need 3 classes:
    - Node class
        - Base class with a virtual void draw and update functions
    - Transform class
        - Responsible for transformations
    - Geometry class
        - Similar to your PointCloud class
        - Responsible for drawing the objects
- Will create either a Transform or Geometry type object

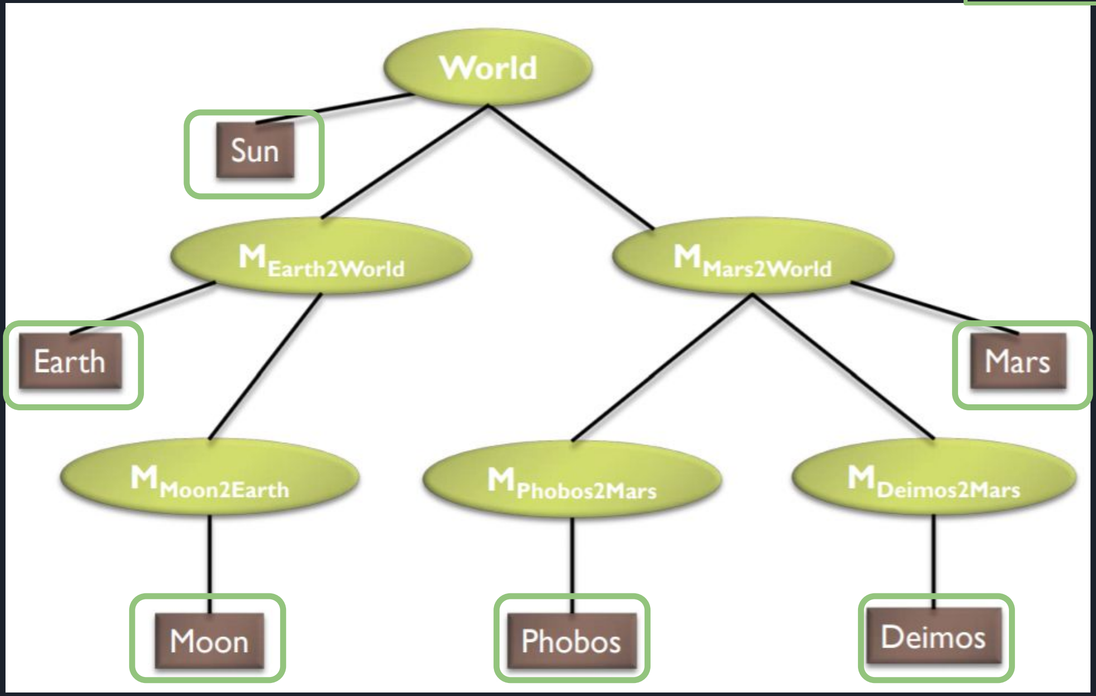
Scene Graph

Transform nodes



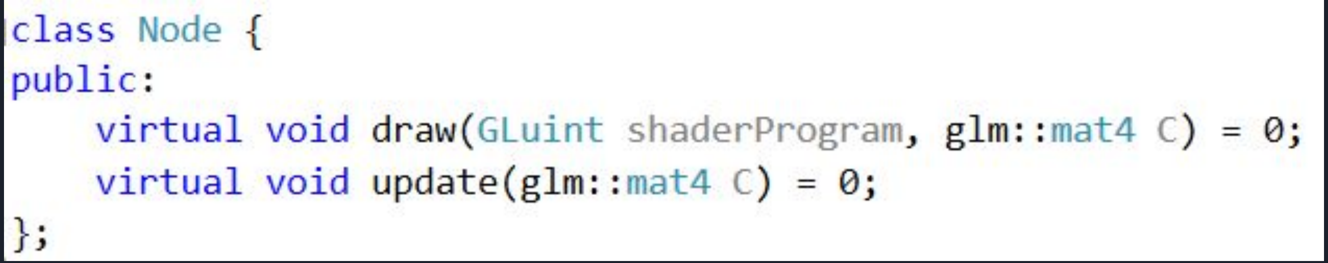
Scene Graph

Geometry nodes



Node Class

- Abstract base class
  - Need to set up the functions that you want both Geometry and Transform classes to have



```
class Node {
public:
    virtual void draw(GLuint shaderProgram, glm::mat4 C) = 0;
    virtual void update(glm::mat4 C) = 0;
};
```

Transform Class

- Derive from Node class
- Functions:
  - draw & update (b/c inheriting from Node)
  - addChild
- Member variables:
  - Transform matrix
    - Matrix that places object relative to parent
  - List of child Nodes

Geometry Class

- Derive from Node class
- Can take straight from PointCloud.cpp
- Functions:
  - draw & update (b/c inheriting from Node)
  - Load, parse… any helper functions you may have had
- Member Variables:
  - model
  - VAO, VBO(s), EBO…
  - Points, normals, indices...

Scene Graph
Building

```
BaseGeo = new Geometry(“base.obj”)

FerrisWheelGeo = new Geometry(“wheel.obj”)

PodGeo = new Geometry("pod.obj")
```

Scene Graph

Building

```
World = new Transform(I)

WheelSpin = new Transform(I)

PodSuspension[] = new Transform(I)

PodSpin[] = new Transform(I)
```

Scene Graph
Building

```
World.addChild(WheelSpin)

WheelSpin.addChild(PodSuspension[])

PodSuspension[].addChild(PodSpin[])
```

Scene Graph
Building

```
World.addChild(BaseGeo)

WheelSpin.addChild(WheelGeo)

PodSpin[].addChild(PodGeo)
```

Scene Graph

Drawing

```
World>draw()
```

Scene Graph
Drawing

- **`World>draw()`**
- Job of Transform’s draw call is to make sure that all its children get drawn
  - Loop through all child nodes
  - Call draw on all child nodes

Scene Graph
Drawing

- Job of Transform’s draw call is to make sure that all its children get drawn **in the correct position**
  - Loop through all child nodes
  - Call draw on all child nodes
- Need to make sure to pass along your transform so the child knows where to go
  - Pass down an updated matrix in the draw function

Scene Graph
Drawing

- Transform draw call:
    - Loop through children
    - Call draw on all children, passing:
        - ShaderProgram
            - So can pass the model matrix to the shader
        - Matrix
            - So we know where to draw the object

Scene Graph

Drawing

- Geometry draw call:
    - Calculate toWorld matrix
        - Based on the passed in matrix and the geometry’s initial model matrix
    - Send that toWorld matrix to the shader
    - glDrawElements(...)

Animation

Animation

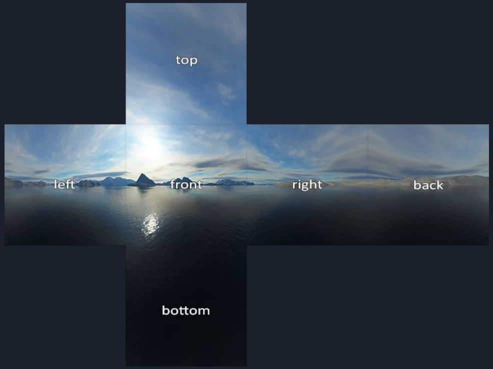
- Need 3 layers of animation independent of each other
- Need to make ride animate
- How?
    - Need to update matrices in transformation nodes
    - Want cyclic motion for linear motion (back and forth, requires direction inversion)
- Where?
    - With the rest of our update calls

Animation

- Where?
  - initialize_objects()
    - Build Ride
  - display_callback()
    - Draw ride by calling draw() on root node (root->draw(...))
    - Animate by calling update functions (root->update(...))

Sky Box

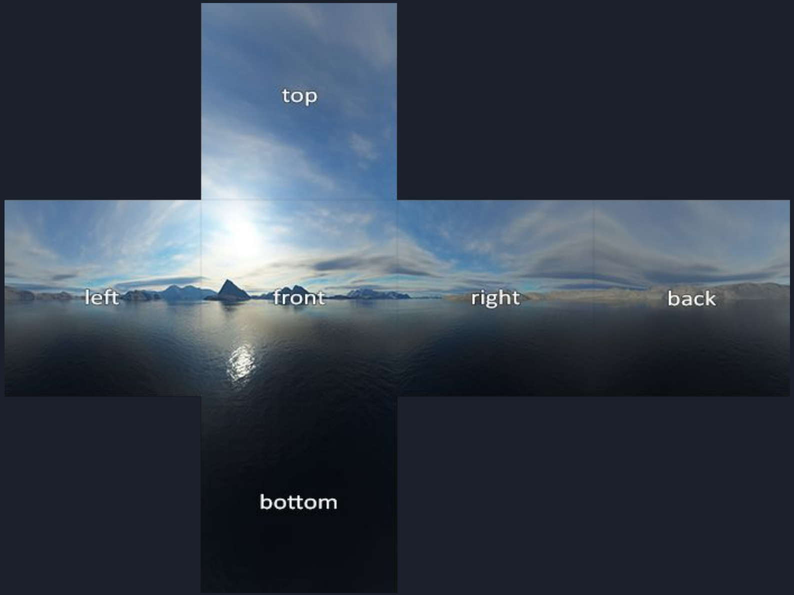
Sky box



- A new set of shaders for sky boxes is needed
- Cube from starter code can be modified and used for skybox
- Tutorial link: https://learnopengl.com/Advanced-OpenGL/Cubemaps

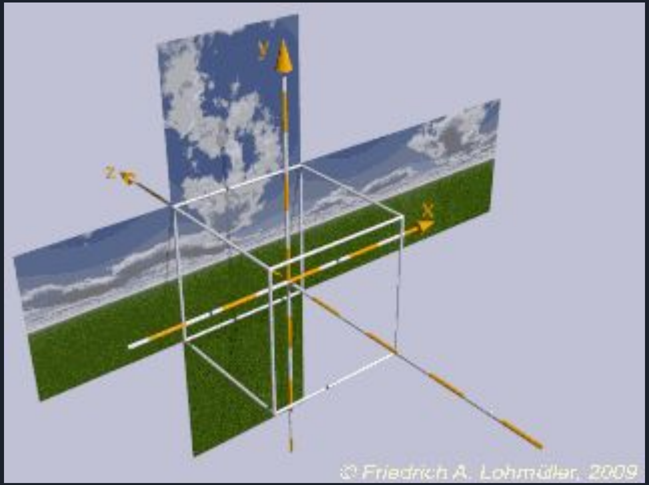
Sky box

- Select your skybox:
  - http://www.f-lohmueller.de/pov_tut/skyboxer/skyboxer_3.htm
  - Create your own high resolution box textures
  - Make sure the orientations are correct as shown on the right



Sky box

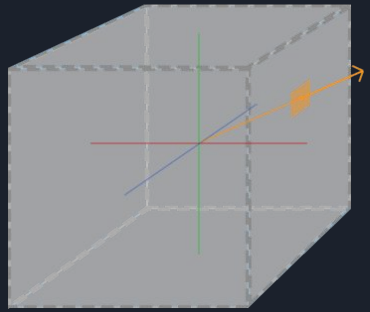
- Set up the cube for the skybox and place the camera inside the cube



http://www.f-lohmueller.de/pov_tut/backgrnd/p_sky9.htm

Sky box

- Coding guide:
  a. Create a cube object. In Skybox.cpp or Cube.cpp, create VAO, VBO and set of vertices just like before.
  b. Create a simple shader program for Skybox,
     - skybox.vert: map input position to texcoords directly.
     - skybox.frag: calculate Fragcolor based on texturecoords using built-in function *texture*.
  c. Create a loadCubemap function to set up 6 textures and return a texture ID.
  d. In the render loop, choose to use the shader program from b. , bind vertex array to the VAO of skybox from a. , and bind GL_TEXTURE_CUBE_MAP to the texture ID created in c.



http://www.f-lohmueller.de/pov_tut/backgrnd/p_sky9.htm

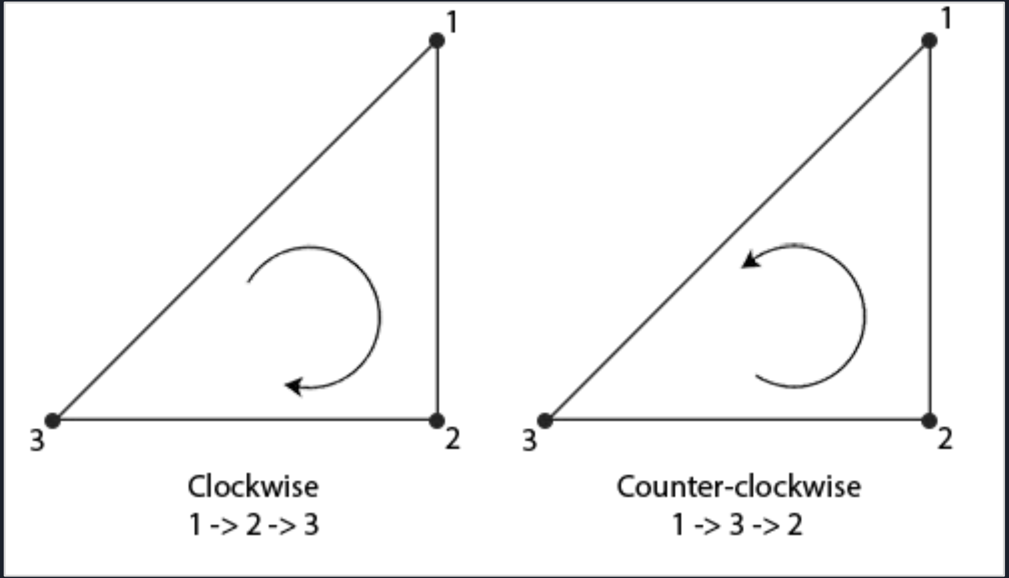
How to render skybox with front face culling

Cube uses counter-clockwise triangles. Here are 2 options to display the inside of the cube as skybox:

1.
   ```
   glEnable(GL_CULL_FACE);
   glCullFace(GL_FRONT);
   ```

2. Telling GL it is defined clockwise:
   ```
   glEnable(GL_CULL_FACE);
   glCullFace(GL_BACK);
   glFrontFace(GL_CW);
   ```

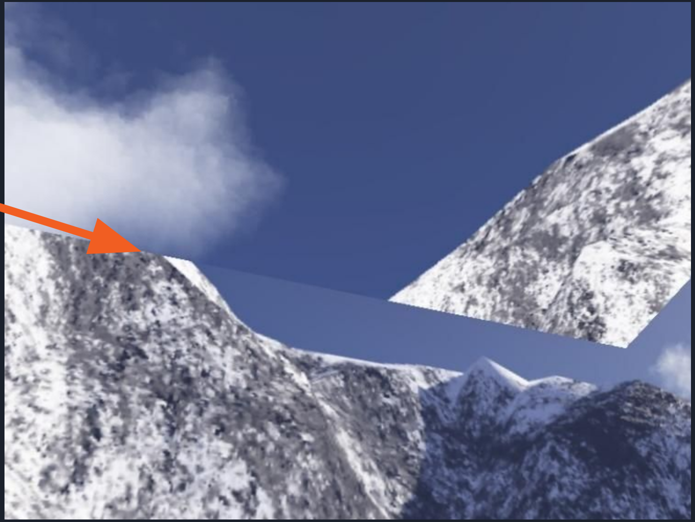
Note: the GL_FRONT and GL_BACK here means the front and the back of a triangle that is being rendered. The front and back is defined by glFrontFace.



Tutorial: https://learnopengl.com/Advanced-OpenGL/Face-culling

Common mistakes

- Wrong texture orientation (mirrored or rotated)
- Discontinuities at edges (see picture on right)
- Incorrect face culling

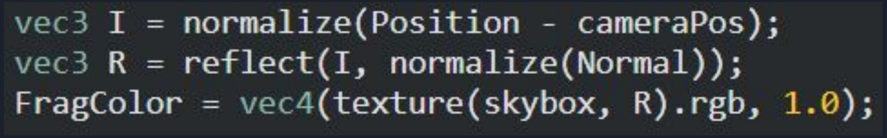


Disco Ball

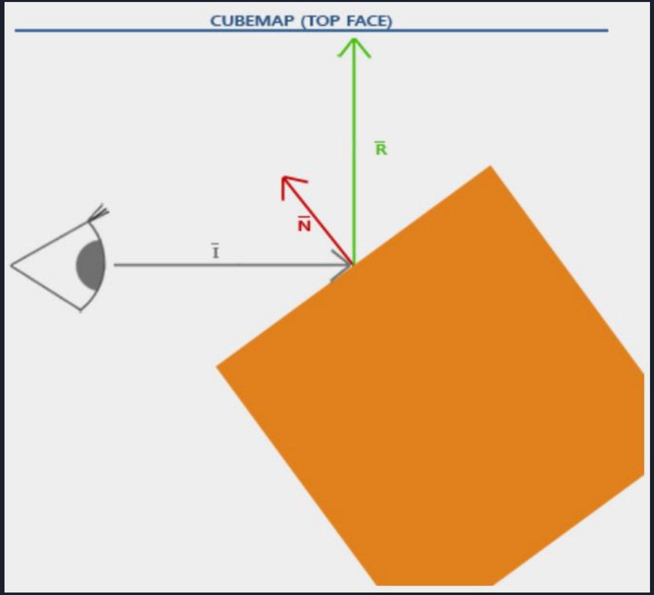
- Mirror reflection effect with low polygon ball model
- Create polygon mesh for ball
- Add environment mapping to shader files shader.vert and shader.frag
- Lighting code is no longer required here
- Tutorial link: https://learnopengl.com/Advanced-OpenGL/Cubemaps

Environment Mapping

- R: reflection vector
- N: normal
- I: view direction
- Calculate reflection vector using GLSL built-in function reflect()



```
vec3 I = normalize(Position - cameraPos);
vec3 R = reflect(I, normalize(Normal));
FragColor = vec4(texture(skybox, R).rgb, 1.0);
```


CUBEMAP (TOP FACE)
$\bar{R}$
$\bar{N}$
$\bar{I}$