Discussion 8
CSE167

Final Project - High-level Description

- Project include:
    - Blog (4 + 3 + 3 = 10 points)
    - Video (5 points)
    - Graphics Applications (85 points)
    - Extra Credit (10 points)

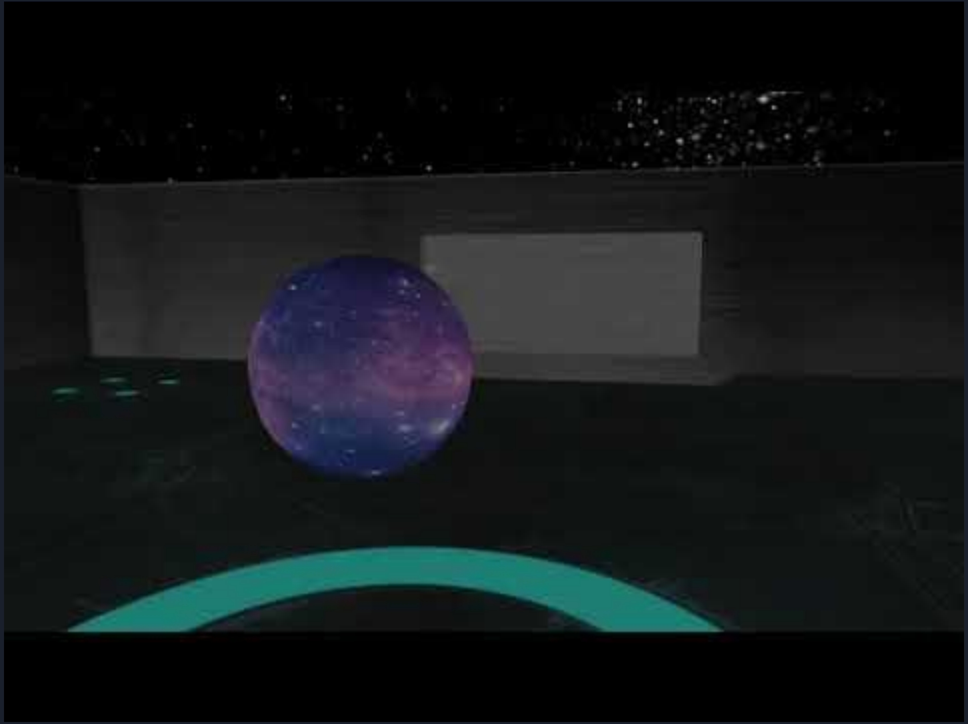
Final Project - Logistics

- Teams of 2 or 3.
- Grading:
    - Technical and creative merits.
    - Time: from 3 to 5:59 pm on December 12, 2019
    - Location:
        - Project grading: basement labs
        - Video Presentation: CSE 1242
    - Keep in mind:
        - 3 skill points per person: any combination of easy (1 pts), medium(2 pts) and hard (3 pts).
        - Maximum of 1 easy point will be counted for each person.
        - First blog should be up by on Wednesday Nov 27th at 11:59 pm. (4 points)

Blog Example

- First blog entry should include:
    - Name of your project
    - Names of your team members
    - 1 paragraph of the content of your project
    - Technical features that you are aiming for
    - Creative efforts
    - Picture (screenshot or sketch, DO NOT copy it from other people’s work!)
- Quite Town

Demo 1 Pong In 3D



Effects:

1. Easy:
    1. Sound effects
    2. Collision Detection
    3. First person camera control with player movements
2. Extra Credit:
    1. Motion blur
    2. Depth of Field
3. Creative Efforts:
    1. Arena selection
    2. Opponent selection

* The levels of difficulty varies from last time the course is offered, the description here matches current requirements.

Demo 2 Reflections to Projections: Lief In a 2D World

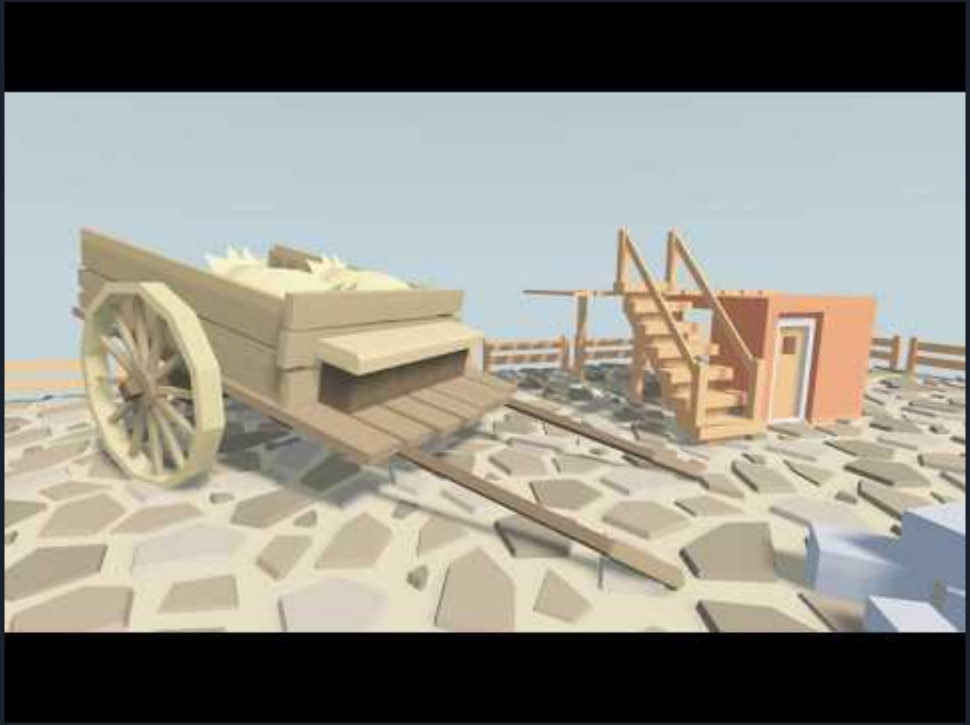
Effects:

1. Easy:
   a. Collision detection
   b. Sound effects
2. Medium:
   a. Procedurally generated terrain
   b. Shadow mapping
   c. Water effect with reflection and refraction
3. Extra credit:
   a. Water effect with reflection of 3D models
4. Creative effort:
   a. Creative content and game design
   b. Player options

Demo 3 Quiet Town

Effects:

1. Easy:
    a. Toon shading
    b. Sound effects
    c. First person camera control with player movements
2. Medium:
    a. Shadow mapping
    b. Procedural cloud
    c. Procedural Ocean with fractal brownian motion based height field
3. Hard:
    a. Fully dynamic god ray with rasterized fragments as occluders
4. Extra credit:
    a. Skylight ambient occlusion



Recommendations on technical difficulties

- Easy: (1 skill point)
    - Toon shading
    - Glow, bloom or halo effect
    - Particle effect
    - Procedurally modeled buildings (no shape grammar)
    - Sound effects
    - Collision detection with bounding spheres or boxes
    - Selection buffer for selection with the mouse
    - First person camera control with player movements

Recommendations on technical difficulties

- Medium: (2 skill points)
    - Bump mapping
    - Surface made with at least two C1 continuous Bezier patches (e.g., flag, water surface, etc.)
    - Procedurally modeled city (no shape grammar)
    - Procedurally generated terrain
    - Procedurally generated plants with L-systems
    - Procedurally modeled buildings with shape grammar
    - Water effect with reflection and refraction
    - Shadow mapping
    - Procedurally generated and animated clouds

Recommendations on technical difficulties

- Hard: (3 skill points)
    - Displacement mapping
    - Screen space post-processed lights
    - Collision detection with arbitrary geometry
    - Shadow Volumes

Recommendations on technical difficulties

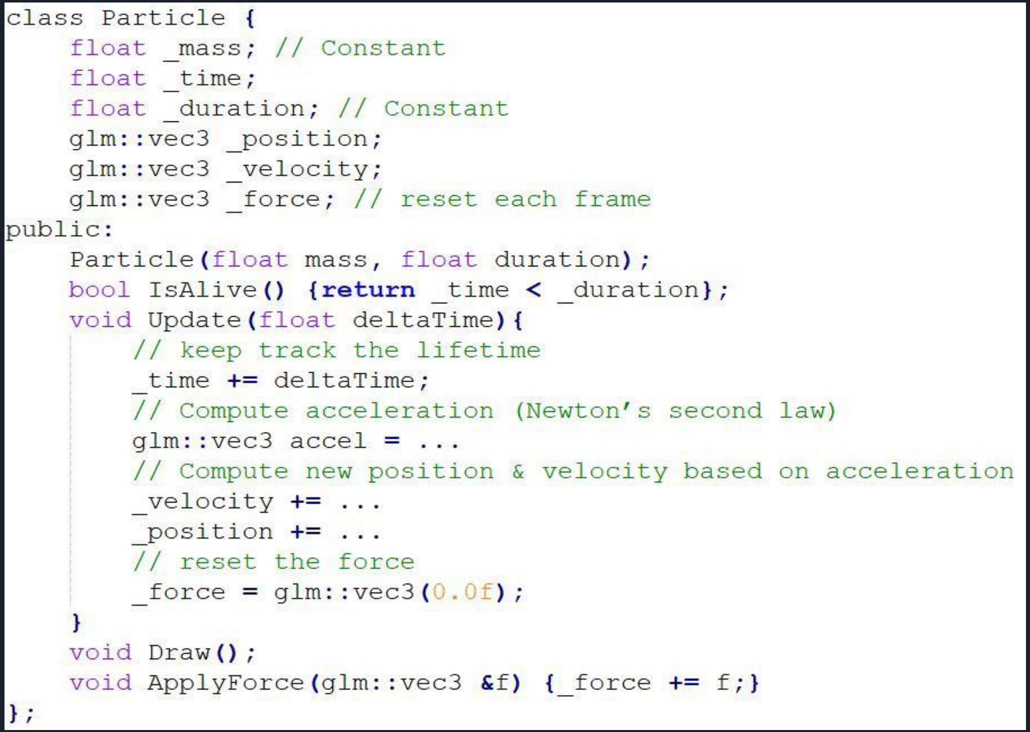
- Extra Credit: (maximum 10 points)
    - Advanced Effects (3 pts each)
        - Water effect with reflection of 3D models (doesn't stack with regular water effect)
        - Screen space ambient occlusion (SSAO) or Screen space directional occlusion (SSDO)
        - Motion blur (tutorial link)
        - Depth of Field
    - Virtual Reality (10 pts)

Particle effect

- Large amount of particles (sprites, points, or anything) follow some combinations of physical and non-physical rules
- At least 200 particles are needed for the final project
- Including two separate stages:
  - Simulation stage:
    - Control spawning and lifetime of particles
    - Apply transformation updates
  - Rendering stage:
    - Update positions of all particles to VBO
    - Use GL_POINT to render or utilize Instanced Rendering if you want to render each particle with geometry other than point

Particle effect

- Simulation stage
- (You don’t have to follow this implementation)



```
class Particle {
    float _mass; // Constant
    float _time;
    float _duration; // Constant
    glm::vec3 _position;
    glm::vec3 _velocity;
    glm::vec3 _force; // reset each frame
public:
    Particle(float mass, float duration);
    bool IsAlive() {return _time < _duration};
    void Update(float deltaTime){
        // keep track the lifetime
        _time += deltaTime;
        // Compute acceleration (Newton’s second law)
        glm::vec3 accel = ...
        // Compute new position & velocity based on acceleration
        _velocity += ...
        _position += ...
        // reset the force
        _force = glm::vec3(0.0f);
    }
    void Draw();
    void ApplyForce(glm::vec3 &f) {_force += f;}
};
```

Particle effect

- Check more

Sound effect

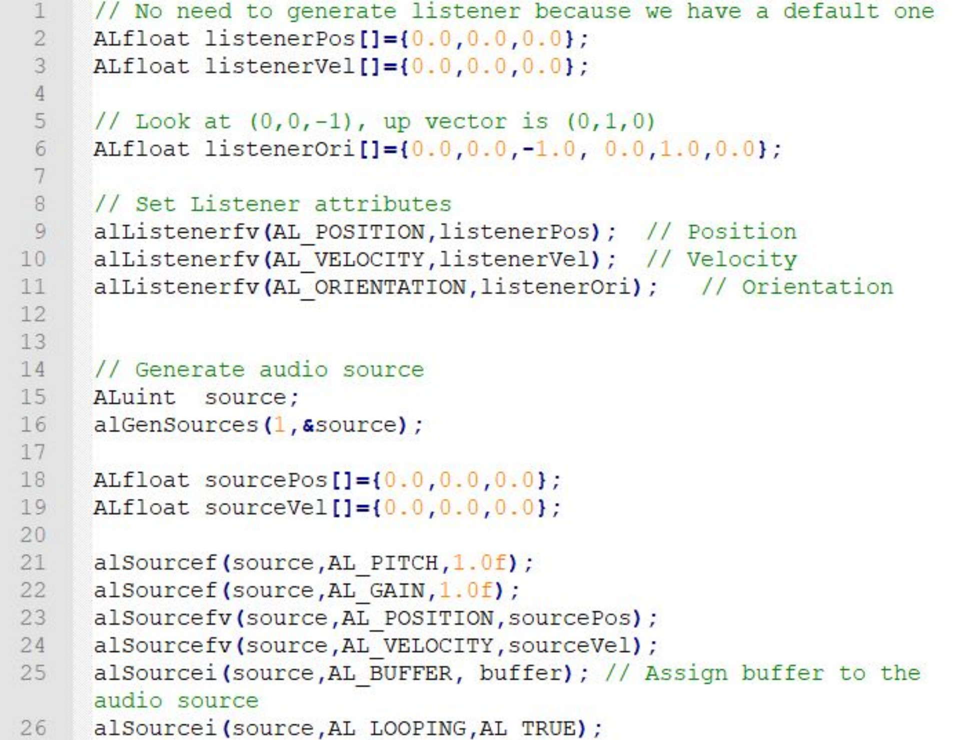
- OpenAL is a pre-approved library for sound effect in the final project
- Requirements:
  - Background music
  - Sounds triggered by events
  - Sounds should be able to play at the same time
- As you might guess, OpenAL's API naming convention follows the OpenGL one
- Create current context and use it during the application lifetime
- "Render" audio using audio context in the "audio scene" (similar to render using opengl context in graphics scene)
- Note: this API deals with audio streams (raw PCM format) instead of with audio codecs.

Sound effect (OpenAL)

- Context: where to play the sound, you can think Window inside of OpenGL
- Listener: OpenAL supports 3D audio, so listener information is very important
- Sources: Information for sound sources
- Buffer: Content responsible for the sound source
- Simple example

Sound effect (OpenAL)

- Code samples to modify listener and audio source properties

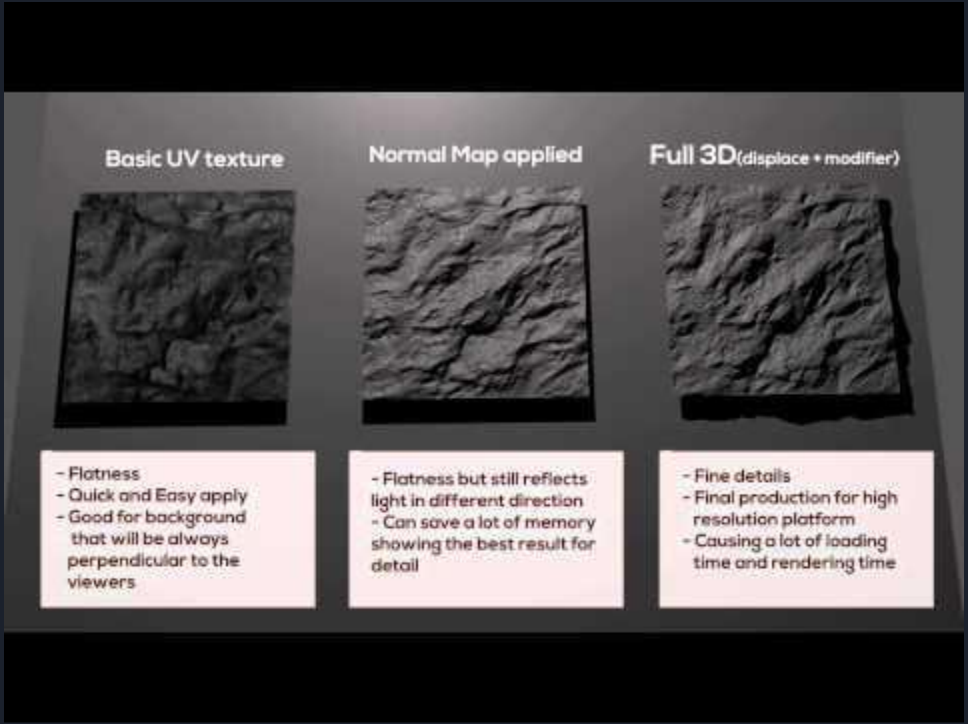


```
// No need to generate listener because we have a default one
ALfloat listenerPos[]={0.0,0.0,0.0};
ALfloat listenerVel[]={0.0,0.0,0.0};

// Look at (0,0,-1), up vector is (0,1,0)
ALfloat listenerOri[]={0.0,0.0,-1.0, 0.0,1.0,0.0};

// Set Listener attributes
alListenerfv(AL_POSITION,listenerPos);  // Position
alListenerfv(AL_VELOCITY,listenerVel);  // Velocity
alListenerfv(AL_ORIENTATION,listenerOri);   // Orientation


// Generate audio source
ALuint  source;
alGenSources(1,&source);

ALfloat sourcePos[]={0.0,0.0,0.0};
ALfloat sourceVel[]={0.0,0.0,0.0};

alSourcef(source,AL_PITCH,1.0f);
alSourcef(source,AL_GAIN,1.0f);
alSourcefv(source,AL_POSITION,sourcePos);
alSourcefv(source,AL_VELOCITY,sourceVel);
alSourcei(source,AL_BUFFER, buffer); // Assign buffer to the audio source
alSourcei(source,AL_LOOPING,AL_TRUE);
```

Bump Mapping

- TBN matrix: A rotation matrix that can transform a vector in tangent space to world space
- Normal map data are in tangent space
- In order to utilize normal map data, you can:
  - Either transform normal map data to world space by multiplying with TBN matrix
  - Or transform light direction, eye position, etc to tangent space by multiplying with transpose(TBN)
- Algorithm to calculate TBN matrix and its derivation: Tutorial

Bump Mapping



Basic UV texture

- Flatness
- Quick and Easy apply
- Good for background that will be always perpendicular to the viewers

Normal Map applied

- Flatness but still reflects light in different direction
- Can save a lot of memory showing the best result for detail

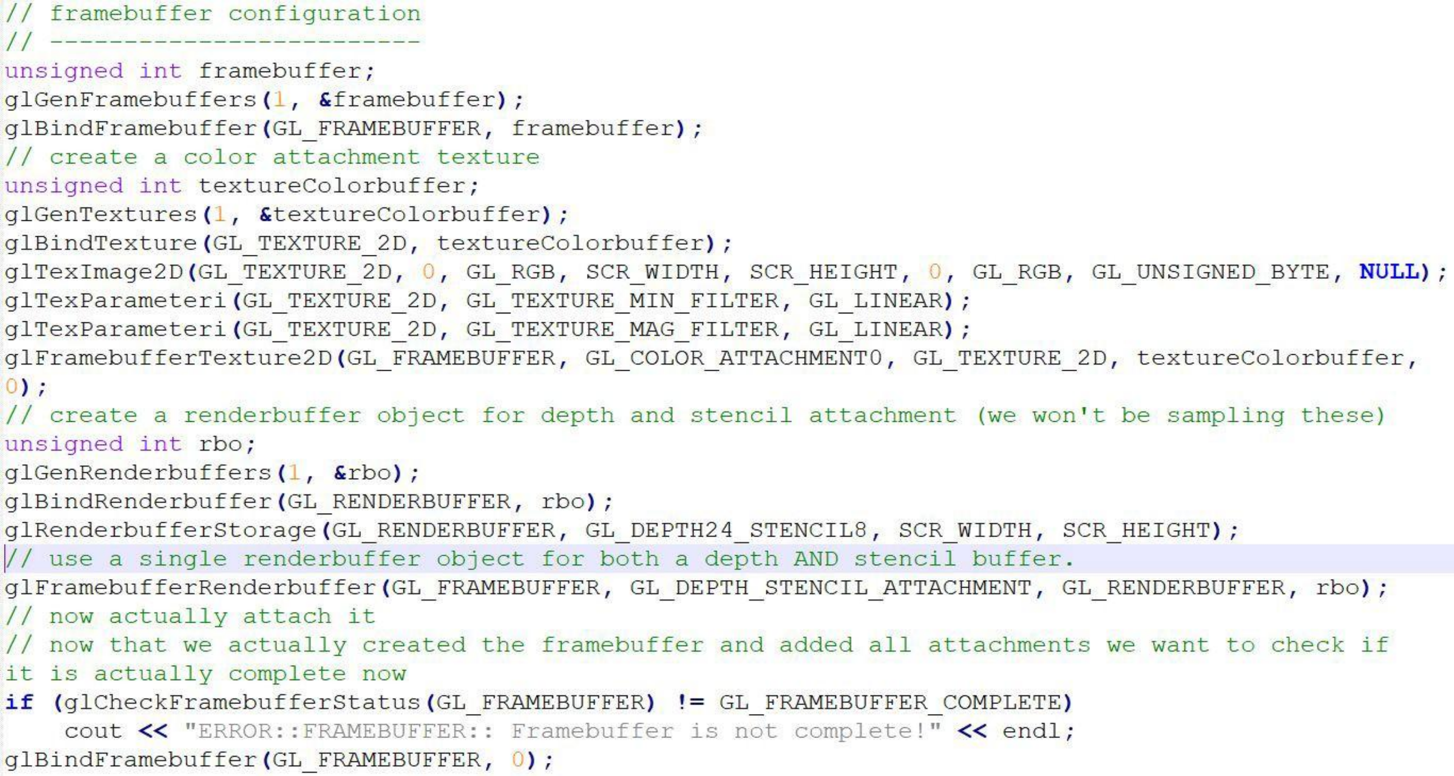
Full 3D(displace + modifier)

- Fine details
- Final production for high resolution platform
- Causing a lot of loading time and rendering time

Framebuffer

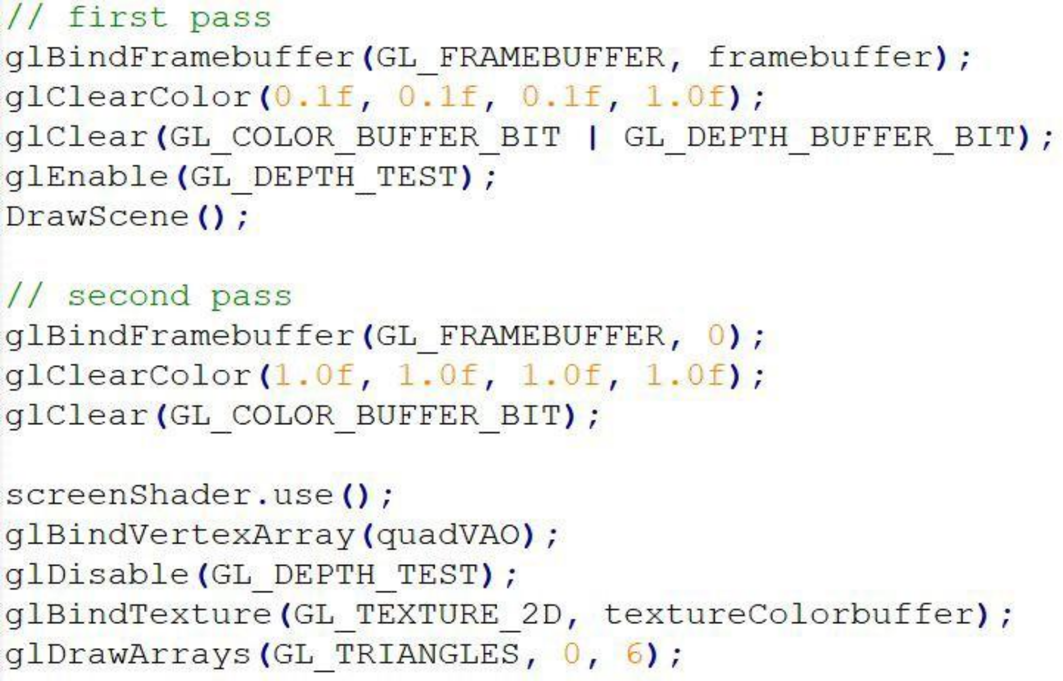
- You will need this for:
  - Shadow
  - Reflection
  - Motion Blur
  - Screen space ambient occlusion
  - Screen space reflection (commonly used in modern game/engine, such as BF5, Unity3D)
  - …
- We may want RGB/normal/depth images from some specific perspectives, and use them later for different graphical effects.
  - Shadow - depth images from the perspective of the light
  - SSAO - screen-space normal image, etc

Framebuffer



```
// framebuffer configuration
// -------------------------
unsigned int framebuffer;
glGenFramebuffers(1, &framebuffer);
glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);
// create a color attachment texture
unsigned int textureColorbuffer;
glGenTextures(1, &textureColorbuffer);
glBindTexture(GL_TEXTURE_2D, textureColorbuffer);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, SCR_WIDTH, SCR_HEIGHT, 0, GL_RGB, GL_UNSIGNED_BYTE, NULL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, textureColorbuffer,
0);
// create a renderbuffer object for depth and stencil attachment (we won't be sampling these)
unsigned int rbo;
glGenRenderbuffers(1, &rbo);
glBindRenderbuffer(GL_RENDERBUFFER, rbo);
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH24_STENCIL8, SCR_WIDTH, SCR_HEIGHT);
// use a single renderbuffer object for both a depth AND stencil buffer.
glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_STENCIL_ATTACHMENT, GL_RENDERBUFFER, rbo);
// now actually attach it
// now that we actually created the framebuffer and added all attachments we want to check if
it is actually complete now
if (glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE)
    cout << "ERROR::FRAMEBUFFER:: Framebuffer is not complete!" << endl;
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

Framebuffer



```
// first pass
glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);
glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glEnable(GL_DEPTH_TEST);
DrawScene();

// second pass
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);

screenShader.use();
glBindVertexArray(quadVAO);
glDisable(GL_DEPTH_TEST);
glBindTexture(GL_TEXTURE_2D, textureColorbuffer);
glDrawArrays(GL_TRIANGLES, 0, 6);
```