CSE 167

Discussion 08 ft. Weichen
11/21/2018

In today's discussion...

- Final Project
- Midterm 2

Announcements

- First blog entry for the final project is due next Tuesday (Nov. 27)
  - Please find some team member(s) as soon as possible if you haven't done yet
- Second blog: Dec. 04
- Midterm 2 is on next Thursday (Nov. 29)
- Project 4 late grading is on next Friday (Nov. 30)
- No class on Thursday
  - Thanksgiving break!

Final Project

- Overview
  - Teams of 2 or 3
  - 3 skill points per person
  - Start early

Final Project

- Grading
  - Blog (10 points) :
    - Nov. 27(4 points)
    - Dec. 04(3 points)
    - Dec. 11(3 points)
  - Video (5 points) : by 3pm on Dec. 13
  - Technical Features (70 points)
  - Creativity (15 points)
  - Extra Credit (10 points):
    - Advanced Effects
    - Virtual Reality

Blog

- The first entry should contain (at a minimum):
  - Name of the project and team members
  - Short description of the project
  - Technical features you are going to implement
  - Creative aspects of your project
- The following entries should be progress updates
  - Progress and any change since the last entry
  - Screenshots

Video

- Important!
- A good/bad demo experience can sometimes boost/dampen users’ impression when they try out your application!
- Note that your grade on the “Creativity” aspects can be affected by your video.

Project Examples

- Projects made by students from last year
- They serve as inspirations but it does not mean all the projects received full credits.
- https://www.youtube.com/playlist?list=PLINx2DKpKpTvFEnpwyzLmtmZK5LXlBP5x

Final Project

- Grading
  - ~~Blog (10 points) :~~
    - Nov. 27(4 points)
    - Dec. 04(3 points)
    - Dec. 11(3 points)
  - ~~Video (5 points) : by 3pm on Dec. 13~~
  - Technical Features (70 points)
  - Creativity (15 points)
  - Extra Credit (10 points):
    - Advanced Effects
    - Virtual Reality

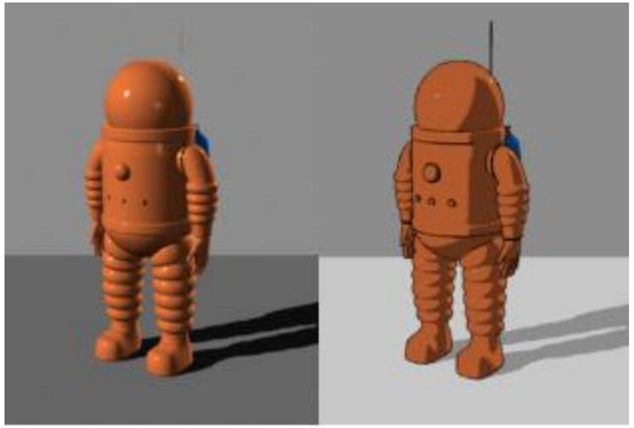
Technical Features

- 3 skill points per person
- Each team must implement at least one medium or hard feature for each team member.
- Some technical features today
  - Toon/Cel shading
  - Particle effect
  - Frame buffer
  - Collision detection

Toon shading

- Designed to make computer graphics appear to be flat by using less shading colors

```
float intensity = dot(lightDir, fragNormal);
if (intensity > 0.95)
    color = ...
else if (intensity > 0.5)
    color = ...
else ...
```


plastic shader toon shader

Technical Features

- Some technical features today
    - ~~Toon/Cel shading~~
    - Particle effect
    - Frame buffer
    - Collision detection

Particle effect

- Large amount of particles (sprites, points, or anything) follow some combinations of physical and non-physical rules
- Simulation stage
- Rendering stage

Particle effect

- Large amount of particles (sprites, points, or anything) follow some combinations of physical and non-physical rules
- Simulation stage
    - Compute all forces acting within the system in the current configuration
    - Compute the resulting acceleration for each particle (a=f/m) and integrate over some small time step (deltaTime) to get new positions
- Rendering stage

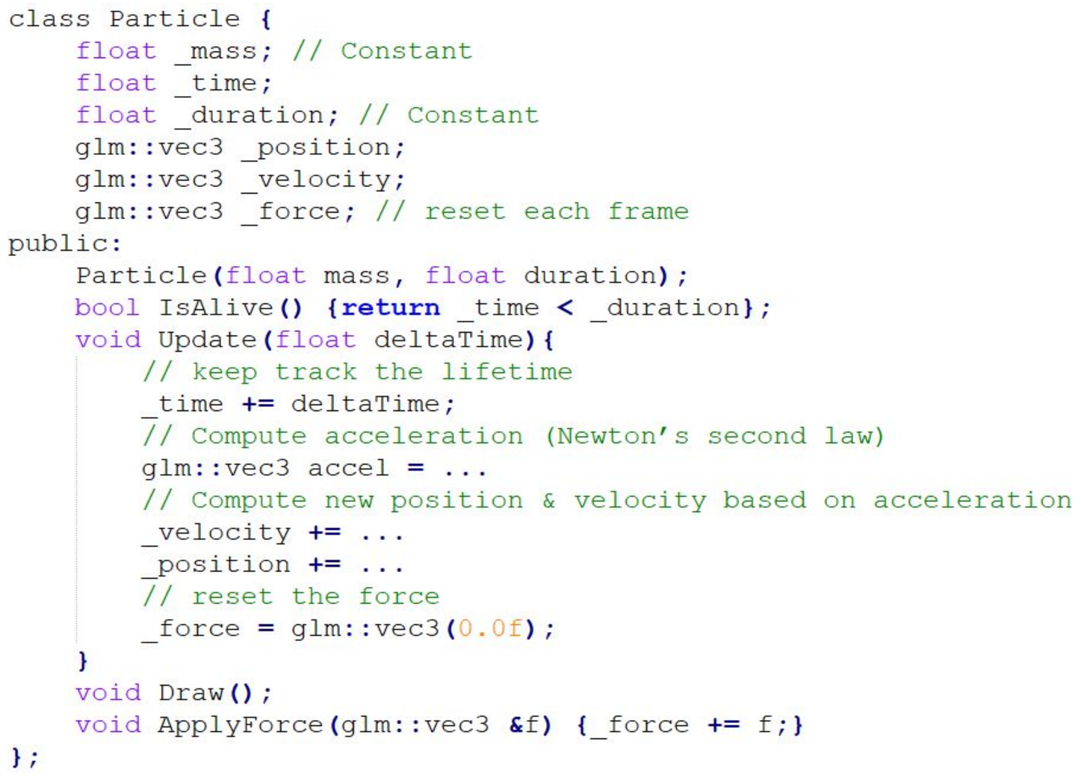
Particle effect

- Large amount of particles (sprites, points, or anything) follow some combinations of physical and non-physical rules
- Simulation stage
- Rendering stage
    - One VBO for the positions of all particles
    - One VBO for the colors of each particle

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;}
};
```

Based on CSE 169 Lecture 10 - Particle Systems

Technical Features

- Some technical features today
    - ~~Toon/Cel shading~~
    - ~~Particle effect~~
    - Frame buffer
    - Collision detection

Frame buffer

- 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)
    - ...

Frame buffer

- 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

Phone
WITH SSAO

I think this GIF is from Half-Life 2

Frame buffer

- Use glGenFrameBuffers() to generate as many frame buffer objects as you need
- Attach a texture to your frame buffer object so you can render to it
- Bind this texture when you want to use the data saved in the texture.

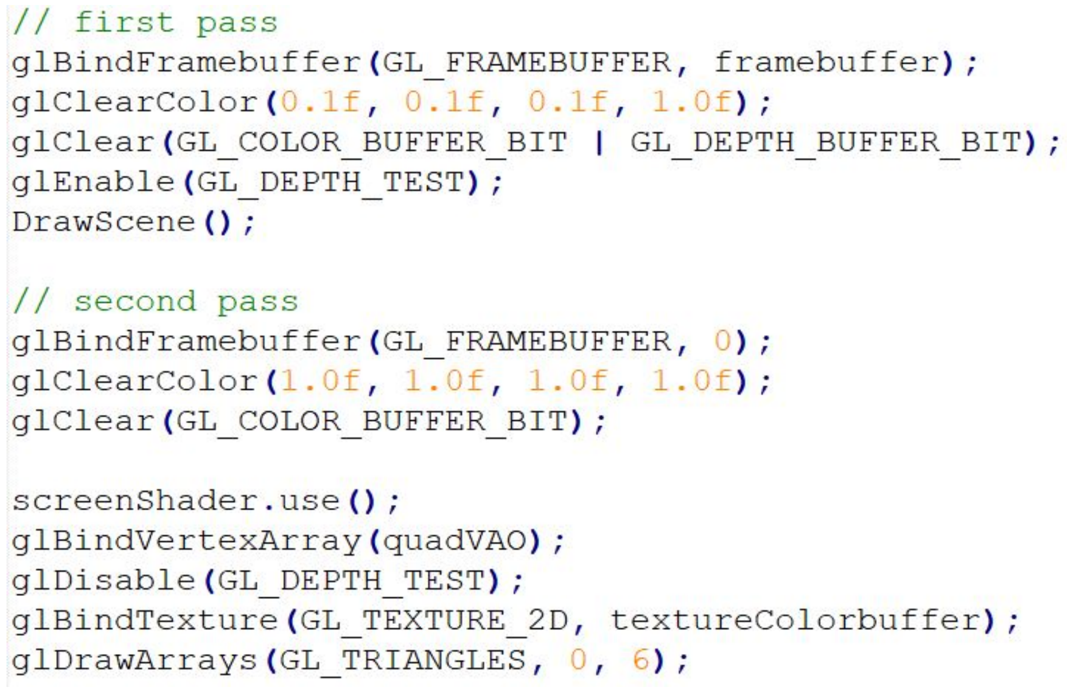
I think this GIF is from Half-Life 2

Frame buffer

```
// 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);
```

Source: https://learnopengl.com/Advanced-OpenGL/Framebuffers

Frame buffer



```
// 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);
```

Source: https://learnopengl.com/Advanced-OpenGL/Framebuffers

Frame buffer

- What if we want RGB and depth (or other) images at the same time?
  - For RGB and depth, bind 2 textures.
  - Use 2 FBO and 2 render passes.
  - glReadPixel(). (Slow)
  - glBlitFramebuffers().

Technical Features

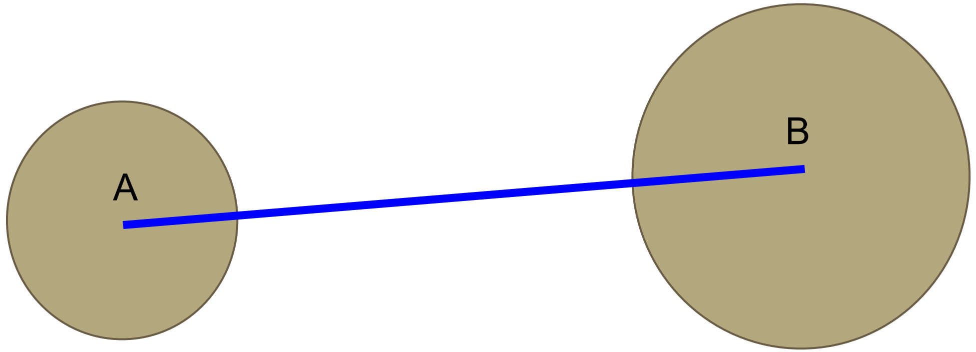
- Some technical features today
    - ~~Toon/Cel shading~~
    - ~~Particle effect~~
    - ~~Frame buffer~~
    - Collision detection

Collision detection

- Bounding spheres
- Bounding boxes
- Arbitrary geometry

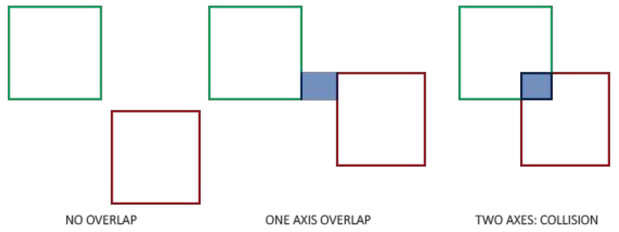
Spheres

- Sphere vs. Sphere
    - Check the distance between the centers of the spheres
        - If the distance is greater than the sum of the spheres’ radii, then they don’t intersect
        - Otherwise, they intersect



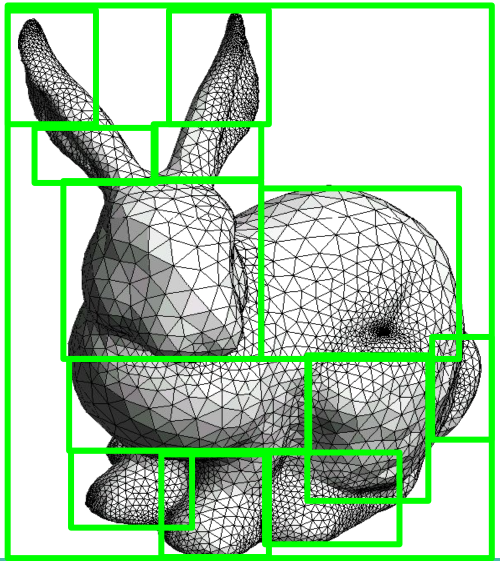
Boxes

- Boxes vs. Sphere: Just check the distance from the sphere's center to each of the box's faces
  - View Frustum Culling
- Boxes vs. Boxes:
  - Axis-aligned Bounding Boxes (AABB):
    - Check overlapping axes
    - 2D example
  - Oriented Bounding Boxes (OBB): Intersection test between triangles
    - Each box only has 12 triangles, so it isn't too hard or time consuming to test
    - You'll need this anyways if you're doing collision detection with arbitrary geometry



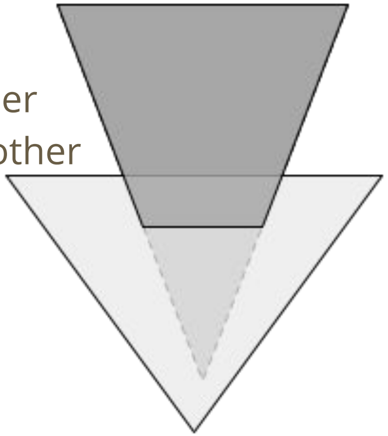
Arbitrary geometry

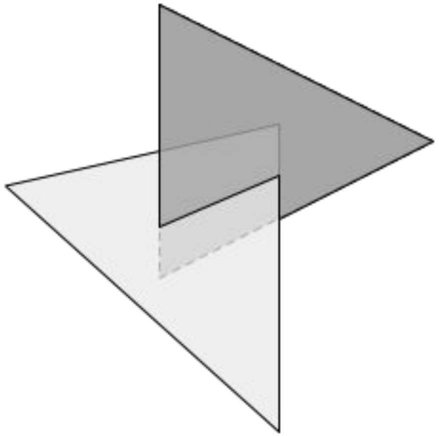
- An intersection test has to be done with every triangle of the 3D object
  - This can be very slow and inefficient if the object has a lot of triangles (for example, the dragon obj file has 871,168 triangles)
- Idea: Break object into multiple sections and only do intersection tests on triangles in a small section instead of whole object
  - Use small bounding spheres or boxes inside the object
  - Check if the intersection returns positive on a sphere or box
    - If no intersection, check next box or sphere
    - If intersection, check for all triangles in box or sphere
- May need scene graphs for recursive intersection tests
  - Good thing you already made a scene graph in project 3
  - You likely need to manually define sizes of the boxes/spheres



Testing triangles

- Once you have determined there is an intersection in the small bounding boxes/spheres, perform intersection tests with the triangles inside them
    - This requires you to iterate through the triangles in both objects and testing for intersection on each pair of triangles
- Only 2 possible cases can happen
    - 2 edges of a triangle intersects the other
    - 1 edge of each triangle intersects the other





The intersection test

- How to do intersection test
  - Determine the planes that each triangle lies on
  - Check if each line segment in a triangle intersects with the other plane
    - If yes, check if the intersection point is within the other triangle
- This can be done relatively quickly since 2 triangles yield a total of 6 test iterations
- We recommend checking one of the following tutorials
  - http://www.applet-magic.com/trintersection.htm
  - http://knight.temple.edu/~lakaemper/courses/cis350_2004/etc/moeller_triangle.pdf
  - **Please be careful of copying + pasting code** - these algorithms weren't written exclusively for an OpenGL context
  - http://web.mst.edu/~chaman/home/pubs/2015WimoTriangleTrianglePublished.pdf
- More information: Google!!!

Midterm 2

- Q&A

Wrap up

- We may talk about more techniques next week
- Enjoy your Thanksgiving break!