Game

EDAF80: Computer Graphics

Rikard Olajos

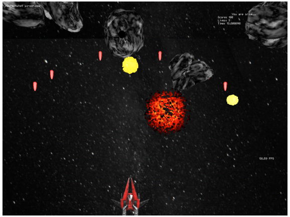
AGENDA

1. Game ideas
2. Collision detection and physics
3. General guidance
4. Assignment 5
5. Going forward

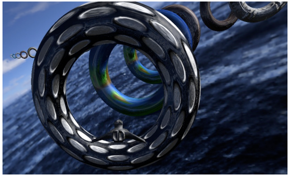
Game ideas

GAME IDEAS

- **Asteroids**
  - Control ship
  - Spawn asteroids randomly
  - Avoid/shoot them down
  - Keep track of health if ship crashes
- **Torus Ride**
  - Place tori along path
  - Control ship
  - Fly-through rings to collect points
  - Time the run
- **Your own idea**
  - Set your creativity free!
  - Discuss with TAs



Azteroidz on YouTube



GENERAL CONSIDERATIONS

- Fixed or dynamic camera?
  - Follow player, or another object?
  - 1st person or 3rd person?
- Manoeuvre by keys (WASD), mouse, or both?
  - Constrained to a plane, or full 3-D?
- Animations
  - Fixed
  - Random
  - Interpolation

ASTEROIDS

- Fixed array of asteroids

```
Node asteroids[N];                // Raw array
std::array<Node, N> asteroids;    // STL array
```

- Respawn when out of view or shot down
- Hide/unhide:

```
if(visible) {
    asteroids[i].render(...);
}
```

- Randomize position, velocity vector, etc.
- Alter appearances using size, shaders, tessellation, noise, ...

TORUS RIDE

- Fixed array of tori

```
Node tori[N];                // Raw array
std::array<Node, N> tori;    // STL array
```

- Fixed or infinite (respawn) path
- Hide/unhide:

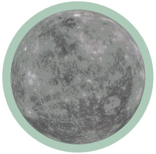
```
if(visible) {
    tori[i].render(...);
}
```

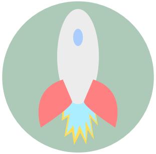
- Place tori along random spline
- Alter appearances using size, rotation, spin, shaders, tessellation, ...

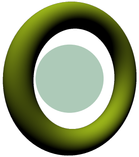
Collision detection and physics

COLLISION DETECTION

- Use ***bounding spheres*** (BS) and perform ***sphere–sphere*** or ***ray–sphere*** collision tests
  - Cheap tests
  - Avoid other primitives





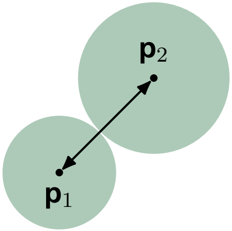


- ***Note***: no need to use an actual sphere – just ***position + radius***
- More types of intersections at realtimerendering.com

SPHERE–SPHERE

- Intersection if

$$|\mathbf{p}_1 - \mathbf{p}_2| < r_1 + r_2$$

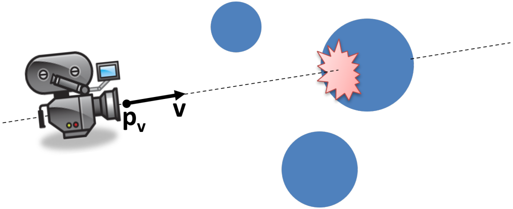


```
bool testSphereSphere(p1, r1, p2, r2);
```

RAY SHOOTING

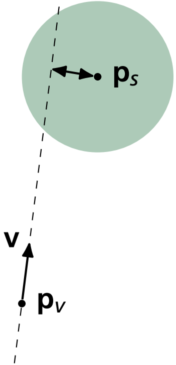
- Ray origin $\mathbf{p}_v$, unit direction $\mathbf{v}$
- "Shoot" ray from camera

```
pv = mCamera.mWorld.GetTranslation();
 v = mCamera.mWorld.GetFront();
```



RAY–SPHERE

- Ray origin $\mathbf{p}_v$, unit direction $\mathbf{v}$
- Sphere at $\mathbf{p}_s$, radius r
- Intersection if
  - $|\text{rejection}(\mathbf{p}_s - \mathbf{p}_v, \mathbf{v})| < r$
  - $\text{rejection}(\mathbf{u}, \mathbf{v}) = \mathbf{u} - \mathbf{v}(\mathbf{u} \cdot \mathbf{v})$



```
bool testRaySphere(pv, v, ps, r);
```

CODE SKETCH

- Spaceship and its BS radius:

```
Node ship;
float ship_BS_radius;
```

- Asteroid and radii lists:

```
Node asteroids[N];
float asteroid_BS_radii[N];
```

- Each frame, test spaceship against all asteroids:

```
for (int i = 0; i < N; i++) {
    if (testSphereSphere(worldPosition(ship),
                         ship_BS_radius,
                         worldPosition(asteroids[i]),
                         asteroid_BS_radii[i])) {
        /* Change heatlh, end game, gain points... */
    }
}
```

PHYSICS: ACCELERATION / INERTIA

- Use fixed ***acceleration*** instead of fixed ***velocity***
  - Smooth starts and stops

```
/* Position and velocity of an object */
vec3 pos = vec3(0.0f, 0.0f, 0.0f);
vec3 vel = vec3(0.0f, 0.0f, 0.0f);

while (!glfwWindowShouldClose(window)) {
    auto const nowTime = (...) now();
    auto const deltaTimeUs = (...) nowTime - lastTime;
    lastTime = nowTime;

    /* Input events */
    // Set some acceleration 'acc' depending on input
    // Add gravity?

    /* Physics */
    float dt = std::chrono::duration<float>(deltaTimeUs).count();
    vel += acc * dt;
    pos += vel * dt;

    /* Render */
    ...
}
```

PHYSICS: ACCELERATION / INERTIA

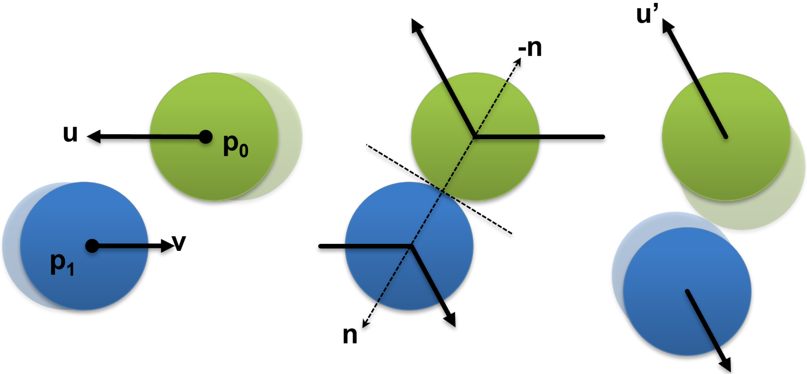
- Use fixed *acceleration* instead of fixed *velocity*
  - Smooth starts and stops

```
/* Position and velocity of an object */
vec3 pos = vec3(0.0f, 0.0f, 0.0f);
vec3 vel = vec3(0.0f, 0.0f, 0.0f);

while (!glfwWindowShouldClose(window)) {
    auto const nowTime = (...) now();
    auto const deltaTimeUs = (...) nowTime - lastTime;
    lastTime = nowTime;

    /* Input events */
    // Set some acceleration 'acc' depending on input
    // Add gravity?

    /* Physics */
    float dt = std::chrono::duration<float>(deltaTimeUs).count();
    vel += acc * dt;
    pos += vel * dt;

    /* Render */
    ...
}
```

- Read more here

ELASTIC COLLISION

- Reflect trajectories along collision normal
- $\mathbf{n} = \text{normalize}(\mathbf{p}_1 - \mathbf{p}_0)$
- $\mathbf{u}' = \text{reflect}(\mathbf{u}, -\mathbf{n})$
- $\mathbf{v}' = \text{reflect}(\mathbf{v}, \mathbf{n})$



General guidance

GENERAL GUIDANCE

- *Keep it simple*: start out with basic features, shaders, etc.
  - Add complexity progressively
  - Total time consumption equivalent to a normal lab
- Reuse your achievements from assignments 1 – 4

GAME STATE

```
enum State {
    NEW_GAME, PLAY_GAME, END_GAME,
};

State current_state = NEW_GAME;

while (!glfwWindowShouldClose(window)) {
    switch (current_state) {
    case NEW_GAME:
        // Do first time setup of variables here
        // Prepare for a new round
        current_state = PLAY_GAME;
        break;
    case PLAY_GAME:
        // Game logic here
        // Control input, physics update, render
        if (player_dead) {
            current_state = END_GAME;
        }
        break;
    case END_GAME:
        // Deal with showing high-scores
        // Ask if the player wants to restart
        if (restart) {
            current_state = NEW_GAME;
        }
    }
}
```

CREATING NEW FILES

- Look in `src/EDAF80/CMakeLists.txt`
- Add the new file names to the `EDAF80_Assignment5` target

```
# Assignment 5
add_executable (EDAF80_Assignment5)
target_sources (
    EDAF80_Assignment5
    PRIVATE
        [[assignment5.hpp]]
        [[assignment5.cpp]]
        [[   new file    ]]
)
target_link_libraries (
    EDAF80_Assignment5
    PRIVATE assignment_setup # Link more libraries here
)
copy_dlls (EDAF80_Assignment5 "{CMAKE_CURRENT_BINARY_DIR}")
```

CREATING NEW FILES

- Look in `src/EDAF80/CMakeLists.txt`
- Add the new file names to the `EDAF80_Assignment5` target

```
# Assignment 5
add_executable (EDAF80_Assignment5)
target_sources (
    EDAF80_Assignment5
    PRIVATE
        [[assignment5.hpp]]
        [[assignment5.cpp]]
        [[   new file    ]]
)
target_link_libraries (
    EDAF80_Assignment5
    PRIVATE assignment_setup # Link more libraries here
)
copy_dlls (EDAF80_Assignment5 "{CMAKE_CURRENT_BINARY_DIR}")
```

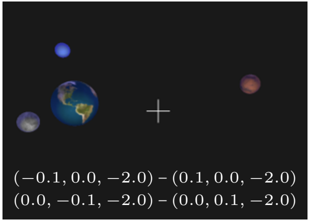
- In Visual Studio: Add new files inside Visual Studio
- For other IDEs: Create files manually
- Rebuild project

IMPORTING NEW MODELS

- Use `bonobo::loadObjects(filename)` in `src/core/helpers.hpp`
  - `filename` is relative to `res/scenes` folder
  - Returns a vector of `bonobo::mesh_data`
  - Other functions, in `parametric_shapes.cpp`, only returned ***one*** instance

DRAWING LINES

- Create `vertexArray` describing the line segments
- Set `mesh_data::drawing_mode` to `GL_LINES` Change line width with
    - `glLineWidth(GLFloat width)`
    - OpenGL documentation
- Crosshair, “laser”, other line effects...
- Consider in which space you render: screen space, world space...



CUBE MAP

- Big sphere as environment
  - Position around the scene, or the camera
  - Disable culling: `glDisable(GL_CULL_FACE);`
- Use for reflections

KEYBOARD EVENTS

```
while (!glfwWindowShouldClose(window)) {
    ...

    /* Input events */
    auto& io = ImGui::GetIO();
    inputHandler.SetUICapture(io.WantCaptureMouse, io.WantCaptureKeyboard);

    glfwPollEvents();
    inputHandler.Advance();
    mCamera.Update(deltaTimeUs, inputHandler);

    if (inputHandler.GetKeycodeState(GLFW_KEY_A) & JUST_PRESSED) {
        // Do something
    }

    /* Game logic & Physics */
    ...

    /* Render */
    ...
}
```

KEYBOARD EVENTS

```
while (!glfwWindowShouldClose(window)) {
    ...

    /* Input events */
    auto& io = ImGui::GetIO();
    inputHandler.SetUICapture(io.WantCaptureMouse, io.WantCaptureKeyboard);

    glfwPollEvents();
    inputHandler.Advance();
    mCamera.Update(deltaTimeUs, inputHandler);

    if (inputHandler.GetKeycodeState(GLFW_KEY_A) & JUST_PRESSED) {
        // Do something
    }

    /* Game logic & Physics */
    ...

    /* Render */
    ...
}
```

- If you want more control: GLFW Documentation

MOUSE EVENTS

```
while (!glfwWindowShouldClose(window)) {
    ...

    /* Input events */
    auto& io = ImGui::GetIO();
    inputHandler.SetUICapture(io.WantCaptureMouse, io.WantCaptureKeyboard);

    glfwPollEvents();
    inputHandler.Advance();
    mCamera.Update(deltaTimeUs, inputHandler);

    glm::vec2 mousePos = inputHandler.GetMousePosition();

    /* Game logic & Physics */
    ...

    /* Render */
    ...
}
```

MOUSE EVENTS

```
while (!glfwWindowShouldClose(window)) {
    ...

    /* Input events */
    auto& io = ImGui::GetIO();
    inputHandler.SetUICapture(io.WantCaptureMouse, io.WantCaptureKeyboard);

    glfwPollEvents();
    inputHandler.Advance();
    mCamera.Update(deltaTimeUs, inputHandler);

    glm::vec2 mousePos = inputHandler.GetMousePosition();

    /* Game logic & Physics */
    ...

    /* Render */
    ...
}
```

- See `FPSCamera::Update()` in `src/core/FPSCamera.inl` for more details

OUTPUT

- Give player feedback through outputs
  - Health, points, game states
- Print to console (`printf` or `std::cout`)
- Or even better, use ImGUI
- Look at the already set up variables for guidance

OUTPUT

- Give player feedback through outputs
  - Health, points, game states
- Print to console (`printf` or `std::cout`)
- Or even better, use ImGUI
- Look at the already set up variables for guidance
- Or even even better, use some textures
  - Create a texture for a game-over state
  - Present on a big quad to the player

RANDOMIZATION

- `int rand(void)`:
  - pseudo-random integral number between `0` and `RAND_MAX`

```
#include <stdlib.h>

int a = rand();                          // [0, RAND_MAX]
float b = rand() / (RAND_MAX + 1.0f);    // [0, 1)
```

RANDOMIZATION

- `int rand(void)`:
  - pseudo-random integral number between 0 and `RAND_MAX`

```
#include <stdlib.h>

int a = rand();                                  // [0, RAND_MAX]
float b = rand() / (RAND_MAX + 1.0f);    // [0, 1)
```

- Set seed with `srand(unsigned int seed);`

DISTRIBUTING YOUR GAME

- Make a folder and include the following:
  - The executable, `EDAF80_Assignment5.exe` in `build/x64-Debug/src/EDAF80`
  - The `shaders` folder
  - The `res` folder
  - The `assimp` DLL (found in the executable folder)
    - `assimp-vc143-mt.dll`
- In the `shaders` and `res` folders, only include files that you use (but keep the correct hierarchy)
- Zip the folder and share!

Assignment 5

ASSIGNMENT 5

- Minimum requirements (Asteroids, Torus Ride)
  - Ship/camera manoeuvrability
  - Use of tessellated objects with shaders
  - Translational and rotational animation
  - Fixed object array (respawn if needed)
  - Game presentation at lab session and on forum gallery
- Optional
  - Game states
  - Collision detection
  - Physics simulation
  - Score count
- Own idea
  - Discuss with TAs

GALLERY

EDAF80: Game Gallery

WHEN YOU ARE DONE

- Make a short post on the forum, `#end-game-gallery`, presenting your game
  - Title
  - Creators
  - Game objectives
  - Features and how you implemented them
  - Screenshots (or a short video)

WHEN YOU ARE DONE

- Make a short post on the forum, `#end-game-gallery`, presenting your game
  - Title
  - Creators
  - Game objectives
  - Features and how you implemented them
  - Screenshots (or a short video)

Good Luck and Have Fun!

Seminar Exercise 5-1: Fragment Shader Art

1. *uv* holds the screen-space coordinates adjusted for the aspect ratio. Visualize this by setting the ***color*** to the euclidian distance to the origin. Use `length()`. You can save the distance in the `float` called `d`.
2. Having just one color can be a bit boring, so let's use a palette. Send the calculated distance to the `palette()` function and use the return value as color.
3. Now let's transform `d` to be something more that just the euclidian distance. Make sure to do the transformations after calculating the palette color.
   - Create concentric sine waves with: `d = sin(d * 8.0 - time) / 8.0;`
   - They are a little dark so boost the values: `d = 0.02 / d;`
   - Negative values don't help us very much so we can use them to double the frequency: `d = abs(d);`
4. Let's add some latitudal and longitudal dependencies as well. Introduce two new variables:
   - `float s = sin(uv.x * 4.0 - time);`
   - `float t = sin(uv.y * 36.0);`
   - Add them to the color calculation: `color *= d + s + t;`
5. Play around with the values and introduce new effects!

Going forward

GOING FORWARD

- Much more to learn at `https://learnopengl.com/`
  - Instancing
  - Advanced lighting
  - Shadows
  - Post-processing
- EDAN35: High-Performance Computer Graphics
- Building your own OpenGL applications
  - C/C++
  - Web-based with Emscripten
    - `emcc`
    - JavaScript
    - `https://webgl2fundamentals.org/`