IO and the Observer Pattern
Announcements and outlook

Guest lecturers (all broadcasted to the classroom):
Oct 20 - SkyBox (Rendering)
Oct 27 - Nvidia (Raytracing, incl. Neural Networks)
Nov 17 - Charm Games (VR games)

Face-to-face grading A1: register! (if selected at ‘random’)
https://docs.google.com/spreadsheets/d/1CMZsU9mmMlj36xcbobDn5N1AOUHDTX9BO14BtYw4hio/edit?usp=sharing

Next deadline: M1

Be prepared for team presentations (like the oral pitch) and M1 face-to-face grading

Assignment A2 posted
Feature clarifications

• **New Organization feature (worth 10 points, take is serious)**

• **Particle effects (basic)**
  
  • Create particle locations and their motion on the CPU (smoke, fire, dirt…)
  
  • Render one Quad at every particle location
  
  • Create a shader (similar to light-up of the salmon that renders the particle in local object coordinates; can also be a texture)

  • `glDrawArraysInstanced` (old technique, no longer used)

• **Advanced particle effects (counts as a new feature)**
  
  • *Use the OpenGL point rendering function instead of quads*
Feature clarification

• **Geometry shader -> Vertex shader (mandatory)**
  
  • *Move vertices around, e.g., deform on collision?*

• Geometry shader (advanced, optional)
  
  • *Create new vertices, e.g., subdivision, explosion, …*
Today

Recap: collisions

Communication between systems:
• The observer pattern

If time permits, some more UI and UX
Recap: Collisions
Motivation: Object selection

- Point inside object boundary?
Motivation: Bullet trajectories

- Line-object or point-object intersection?

https://forum.unity.com/threads/2d-platformer-shooting.365971/
Collision Configurations?

- Segment/Segment Intersection
  - *Point on Segment*

- Polygon inside polygon
Lines & Segments

Segment $\Gamma$ from $p = (x_0, y_0)$ to $q = (x_1, y_1)$

Find the line through $p = (x_0, y_0)$ and $q = (x_1, y_1)$?

- Parametric: $\Gamma(t), t \in (-\infty, \infty)$
- Implicit: $Ax + By + C = 0$
  - Solve 2 equations in 2 unknowns (substitute $A^2 + B^2 = 1$)
Line-Line Intersection

\[\Gamma^1 = \begin{cases} x^1(t) = x^1_0 + (x^1_1 - x^1_0)t & t \in [0,1] \\ y^1(t) = y^1_0 + (y^1_1 - y^1_0)t \end{cases}\]

\[\Gamma^2 = \begin{cases} x^2(r) = x^2_0 + (x^2_1 - x^2_0)r & r \in [0,1] \\ y^2(r) = y^2_0 + (y^2_1 - y^2_0)r \end{cases}\]

Intersection: \(x\) & \(y\) values equal in both representations - two linear equations in two unknowns \((r,t)\)

\[
\begin{align*}
x^1_0 + (x^1_1 - x^1_0)t &= x^2_0 + (x^2_1 - x^2_0)r \\
y^1_0 + (y^1_1 - y^1_0)t &= y^2_0 + (y^2_1 - y^2_0)r
\end{align*}
\]

Question: What is the meaning if the solution gives \(r,t < 0\) or \(r,t > 1\) ?
Bounding Volume Intersection

- Axis aligned bounding box (AABB)
  - \( A.LO \leq B.HI \) && \( A.HI \geq B.LO \) (for both \( X \) and \( Y \))

- Circles
  - \[ ||A.C - B.C|| < A.R + B.R \]

Center

Radius
Different intersection types

*Point – line (which side of line)*

*Point – object (contained)*

*Line – line*

*Line – object*

*Object – object*
Moving objects

• Sweep – test intersections against before/after segment
  • Avoid “jumping through” objects
  • How to do efficiently?

• Boxes?

• Spheres?
Getting stuck – Collision resolution

**Solutions:**

- If collision is detected, stop at position of previous step
  - *Can still cause locked situations since objects can go back in time*

- Only count collision when objects are moving towards each other
  - *Requires to consider motion direction of both objects*
    - What if one is still?
Complexity?

What is the complexity of checking collision between all objects in your game (naïve version)?

A: Linear in number of objects

B: Quadratic

C: Logarithmic

D: Exponential
Hierarchical Bounding Volumes

Bound Bounding Volumes:

- Use (hierarchical) bounding volumes for groups of objects

- How to group boxes?
  - Closest
  - Most jointly compact (how?)
Hierarchical Bounding Volumes

Bind Bounding Volumes:

- Use (hierarchical) bounding volumes for groups of objects

- Challenge: dynamic data…
  - Need to update hierarchy efficiently
Spatial Subdivision DATA STRUCTURES

- Subdivide space (bounding box of the “world”)
- Hierarchical
  - *Subdivide each sub-space (or only non-empty sub-spaces)*
- Lots of methods
  - *Grid, Octree, k-D tree, (BSP tree)*
Regular Grid

**Subdivide space into rectangular grid:**

- Associate every object with the cell(s) that it overlaps with.
- Test collisions only if cells overlap.

In 3D: regular grid of cubes (voxels):
Creating a Regular Grid

**Steps:**

• Find bounding box of scene
• Choose grid resolution in x, y, z
• Insert objects
• Objects that overlap multiple cells get referenced by all cells they overlap
Regular Grid Discussion

**Advantages?**
- Easy to construct
- Easy to traverse

**Disadvantages?**
- May be only sparsely filled
- Geometry may still be clumped
Adaptive Grids

- Subdivide until each cell contains no more than \( n \) elements, or maximum depth \( d \) is reached

Nested Grids

Octree/(Quadtrees)

- This slide is courtesy of Fredo Durand at MIT
Collision Resolution

Today: simplified example

Upcoming lecture:

Physics-based simulation
Basic Particle Simulation (first try)

How to compute the change in velocity?

\[ d_t = t_{i+1} - t_i \]
\[ \vec{v}_{i+1} = \vec{v}_i + \Delta v \]
\[ \vec{p}_{i+1} = \vec{p}(t_i) + \vec{v}_i d_t \]
Particle-Plane Collisions

• Change in direction of normal

Frictionless

\[ \Delta v = 2(v^- \cdot \hat{n})\hat{n} \]

\[ v^+ = v^- + \Delta v \]

Loss of energy

\[ \Delta v = (1 + \epsilon)(v^- \cdot \hat{n})\hat{n} \]
Particle-Particle Collisions (spherical objects)

Before collision

\[ m_1 \quad v_1^- \quad m_2 \quad v_2^- \]

After

\[ m_1 \quad v_1^+ \quad m_2 \quad v_2^+ \]

Response:

\[ v_1^+ = v_1^- - \frac{2m_2}{m_1 + m_2} \frac{(v_1^- - v_2^-) \cdot (p_1 - p_2)}{||p_1 - p_2||^2} (p_1 - p_2) \]

\[ v_2^+ = v_2^- - \frac{2m_1}{m_1 + m_2} \frac{(v_2^- - v_1^-) \cdot (p_2 - p_1)}{||p_2 - p_1||^2} (p_2 - p_1) \]

- This is in terms of velocity
- Upcoming lectures: derivation via impulse and forces
IO and the Observer Pattern
int main(int argc, char* argv[]) {
...
2. Mainloop:
while (!world.is_over()) {
  2.1 Event processing
  2.2 Game state update
  2.3 Rendering a frame
}
...
}
Event Processing

Mouse event, Keyboard event, etc.

Credits:
https://svgsilh.com/image/25711.html
Event Processing: Event Queuing

Mouse event, Keyboard event, etc.

Credits:
https://svgsilh.com/image/25711.html
Event Processing: Event Polling

Mouse event, Keyboard event, etc.

while (!world.is_over()) {

2.1 Event processing

glfwPollEvents();

Credits:
https://svgsilh.com/image/25711.html
Event Processing: Event Callback

GLFW calls corresponding callbacks:

- `void World::on_key(GLFWwindow*, int key, int action, int mod)`
  
  --> You set salmon motion here.

- `void World::on_mouse_move(GLFWwindow* window, double xpos, double ypos)`
  
  --> You set salmon rotation.
How does GLFW know which callback to call?
Event Processing: Event Callback

How does GLFW know which callback to call?

-> Registered in initialization:

world.init(...)  
glfwSetKeyCallback
glfwSetCursorPosCallback
Mainloop

```c
int main(int argc, char* argv[]) {
...

2. Mainloop:
while (!world.is_over()) {
  2.1 Event processing
  2.2 Game state update
  2.3 Rendering a frame
}
...
}
```
glfwPollEvents

**Asynchronous?**

**Reference:**
https://www.glfw.org/docs/3.0/group__window.html#ga37bd57223967b4211d60ca1a0bf3c832

- “This function processes only those events that have already been received and then returns immediately. Processing events will cause the window and input callbacks associated with those events to be called.”

  - synchronous!

- “On some platforms, certain callbacks may be called outside of a call to one of the event processing functions.”

  - asynchronous! :/
Workaround?

• https://stackoverflow.com/questions/36579771/glfw-key-callback-synchronization

Generally speaking, the way that you should be handling input is to keep a list of keys, and record their last input state.

```cpp
struct key_event {
    int key, code, action, modifiers;
    std::chrono::steady_clock::time_point time_of_event;
};
```

```cpp
std::map<int, bool> keys;
std::queue<key_event> unhandled_keys;
void handle_key(GLFWwindow* window, int key, int code, int action, int modifiers) {
    unhandled_keys.emplace_back(key, code, action, modifiers, std::chrono::steady_clock::now);
}
```

Then, in the render loop (or you can separate it into a different loop if you're confident with your multithreading + synchronization abilities) you can write code like this:

```cpp
float now = glfwGetTime();
static float last_update = now;
float delta_time = now - last_update;
last_update = now;
handle_input(delta_time);
```
The Observer Pattern

• **Gang of Four (GoF)**
  • Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides
  • *Design Patterns: Elements of Reusable Object-Oriented Software* (1994)

• **A pattern described by the GoF**
  • event-driven
    • clients register for an event

Good ref (object oriented):
https://gameprogrammingpatterns.com/observer.html
Use Cases

• Rewards

• Communication between systems (in ECS)

• User input

• ???
• Define a common interface
• All observers inherit from that interface
Lambda Functions

Definition:

• auto y = [] (int first, int second) { return first + second; }; 
Call: int z = y(1+3);

• Infers return type for simple functions (single return statement)
  • otherwise
    auto y = [] (int first, int second) -> int { return first + second; }; 

• Can capture variables from the surrounding scope.
  int scale;
  auto y = [] (int first, int second) -> int { return scale*first + second; }; 
  auto y = [&] (int first, int second) -> int { return scale*first + second; };
Observer Pattern – With Functions

• function with matching signature instead of class
A function that accepts a function

• **Using `std::function`**

```cpp
#include<functional>

void LambdaTest (const std::function<void (int)>& f) {
    ...
}
```

• **Using templates**

```cpp
template<typename Func>
void LambdaTest(Func f) {
    f(10);
}
```

• *use templates to accept any argument with an operator()*
Observer Pattern – With Functions

- function with matching signature instead of class

```cpp
std::vector<std::function<void ()>> callbacks

attach(std::function<void ()> fn)
```

```cpp
for each v in views
  v.update()

model.getState();
```
IO in our Template

• ?????????

// Setting callbacks to member functions (that's why the redirect is needed)
// Input is handled using GLFW, for more info see
// http://www.glfw.org/docs/latest/input_guide.html
glfwSetWindowUserPointer(window, this);
auto key_redirect = []((GLFWwindow* wnd, int _0, int _1, int _2, int _3) { ((WorldSystem*)glfwGetWindowUserPointer(wnd))->_on_key(wnd, _0, _1, _2, _3); });
auto cursor_pos_redirect = []((GLFWwindow* wnd, double _0, double _1) { ((WorldSystem*)glfwGetWindowUserPointer(wnd))->_on_mouse_move(wnd, { _0, _1 }); });
glfwSetKeyCallback(window, key_redirect);
glfwSetCursorPosCallback(allcallback(window, cursor_pos_redirect);

• Function signature

GLFWAPI GLFWkeyfun glfwSetKeyCallback(GLFWwindow* window, GLFWkeyfun cbfun);

/**
 * Sets the Unicode character callback.
 */

// This function sets the character callback of the specified window. It is called when a Unicode character is input.

// The character callback is intended for Unicode text input. A character is a sequence of Unicode characters, it is keyboard layout dependent, whereas the keys [key callback](#) is not. Characters are usually related to physical keys, as a key may produce zero, one or more characters.

// Search Online
Performance?

• *Isn’t this slow?*