CPSC 436D: Video Game Programming

Assignment 1 Template Walk-Through

(Spring 2019)

1

...

2

Before We Start…

First Glance: Main

a1.cpp

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

1. Initialization

2. Mainloop:

```
while (!world.is_over()) {…}
```

3. Resource clean-up & Exit

```
}
```

First Glance: Main

a1.cpp

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

1. Initialization

world.init(windows_size)

—-> world.cpp:L41:init

2. Mainloop:

while (!world.is_over()) {…}

3. Resource clean-up & Exit

world.destroy();
return EXIT_SUCCESS;

—-> world.cpp:L126:destroy

}

5

Resource Allocation and Free

world.init(windows_size)

Processes that only need to be done once:

world.destroy()

6

Resource Allocation and Free

world.init(windows_size)

Processes that only need to be done once:

- Window, OpenGL context & objects
- Event callback registration (keyboard, mouse inputs)
- Loading audio (also starts playing)
- Loading and initialize game entities (salmon, water etc.)

world.destroy()

7

Resource Allocation and Free

world.init(windows_size)

Processes that only need to be done once:

- Window, OpenGL context & objects
- Event callback registration (keyboard, mouse inputs)
- Loading audio (also starts playing)
- Loading and initialize game entities (salmon, water etc.)

world.destroy()

Release resources: **To avoid memory leaks**

8

Mainloop

a1.cpp

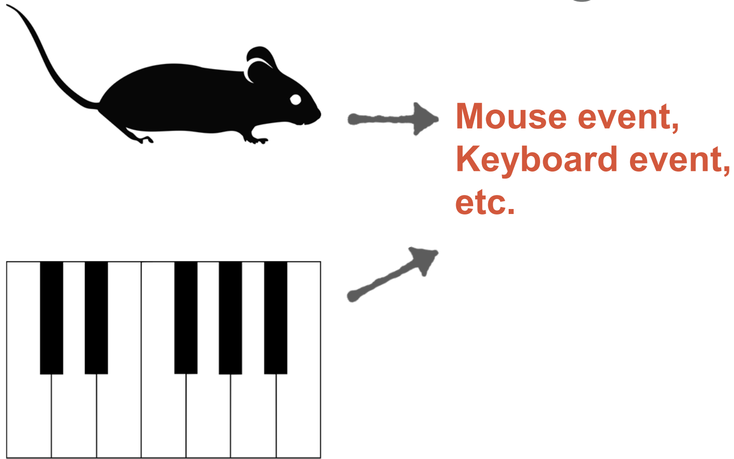
```
int main(int argc, char* argv[]) {
        …
        2. Mainloop:
        while (!world.is_over()) {…}
        …
}
```


Mainloop

a1.cpp

```
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
        }
        …
}
```

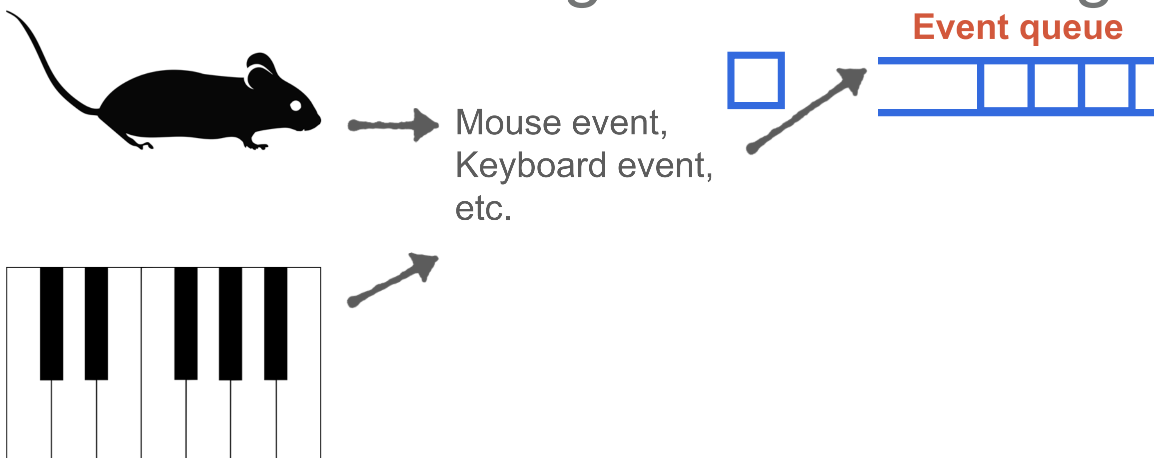
Event Processing



Credits:
https://pixabay.com/en/mouse-mouse-silhouette-lab-mouse-2814846/
https://svgsilh.com/image/25711.html

11

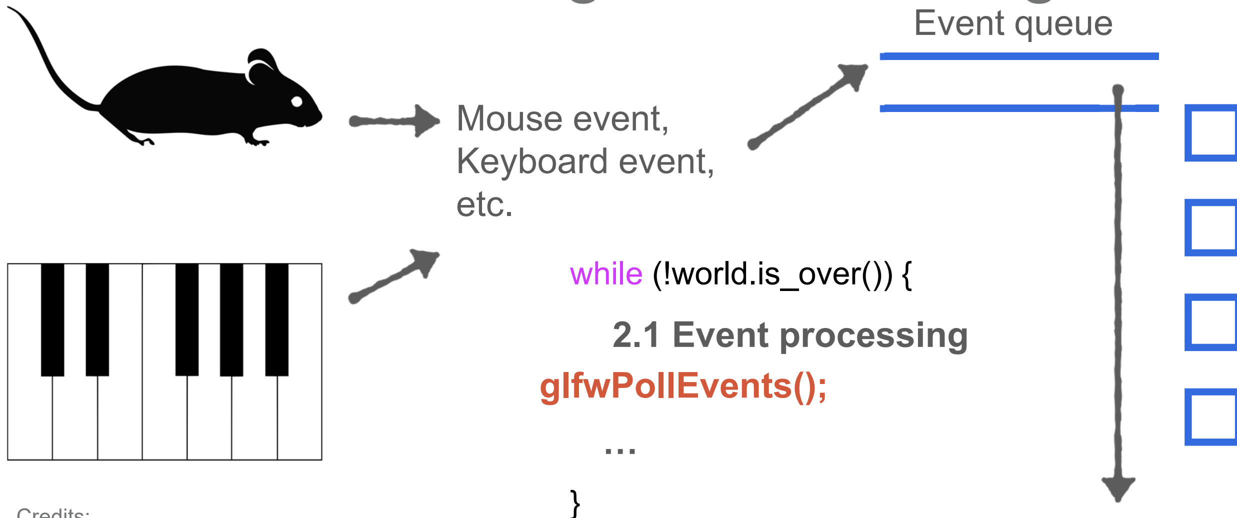
Event Processing: Event Queuing



Credits:
https://pixabay.com/en/mouse-mouse-silhouette-lab-mouse-2814846/
https://svgsilh.com/image/25711.html

12

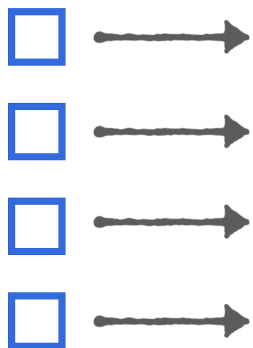
Event Processing: Event Polling



Credits:
https://pixabay.com/en/mouse-mouse-silhouette-lab-mouse-2814846/
https://svgsilh.com/image/25711.html

13

Event Processing: Event Callback

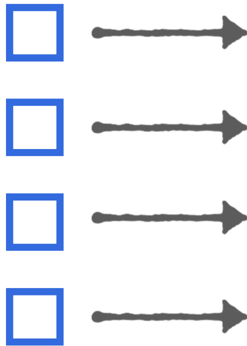


GLFW calls corresponding callbacks:

- void World::on_key(GLFWwindow*, int key, int, int action, int mod)
  —-> world.cpp:L369
  You need to set salmon motion here.
- void World::on_mouse_move(GLFWwindow* window, double xpos, double ypos)
  —-> world.cpp:L399
  You need to fill this function to set salmon rotation.

14

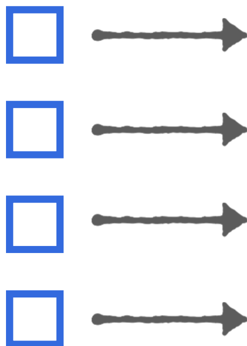
Event Processing: Event Callback



How does GLFW know which callback to call?

15

Event Processing: Event Callback



How does GLFW know which callback to call?

—-> Registered in initialization:

```
world.init(windows_size)
glfwSetKeyCallback
glfwSetCursorPosCallback
```

16

Mainloop

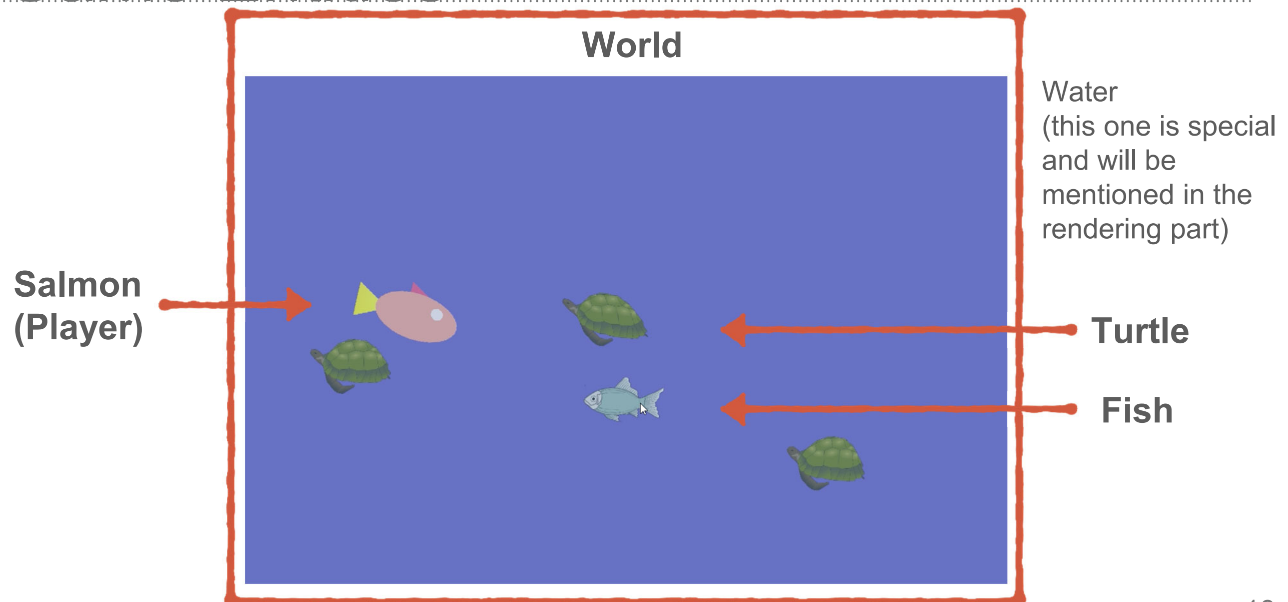
a1.cpp

```
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
        }
        ...
}
```

~~2.1 Event processing~~

17

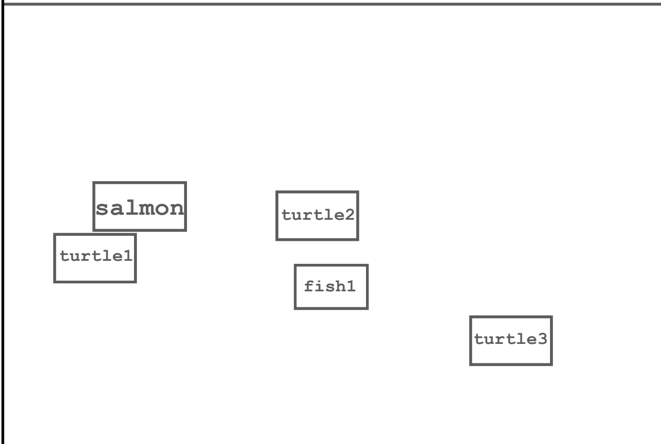
Game Entities



18

Salmon, Turtle, Fish: Two Views

What the game sees:



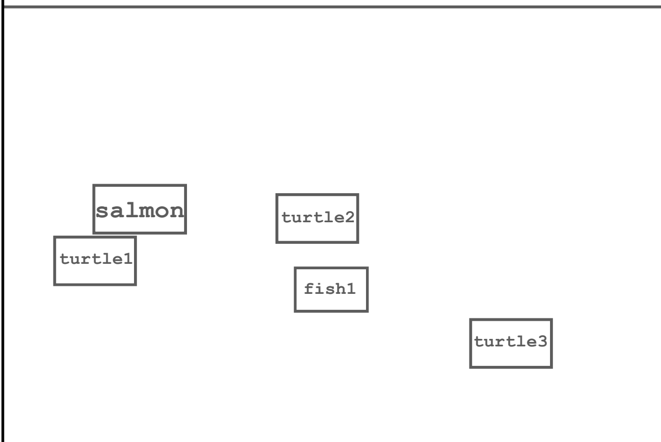
What the player sees:

19

Salmon, Turtle, Fish: Two Views

What the game sees:

(width, height)



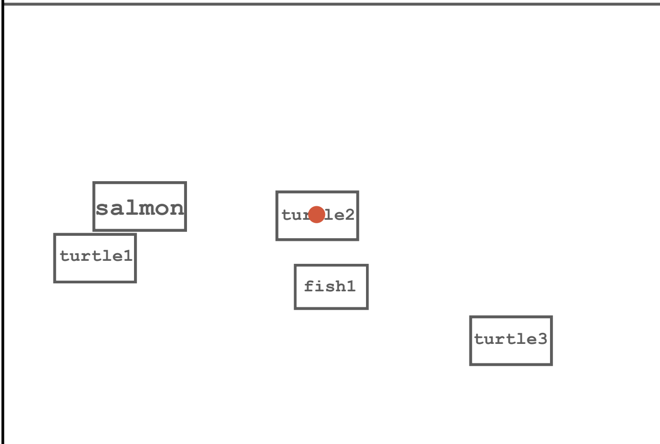
(0, 0)

What the player sees:

20

Salmon, Turtle, Fish: Two Views

What the game sees: **(width, height)** **What the player sees:**

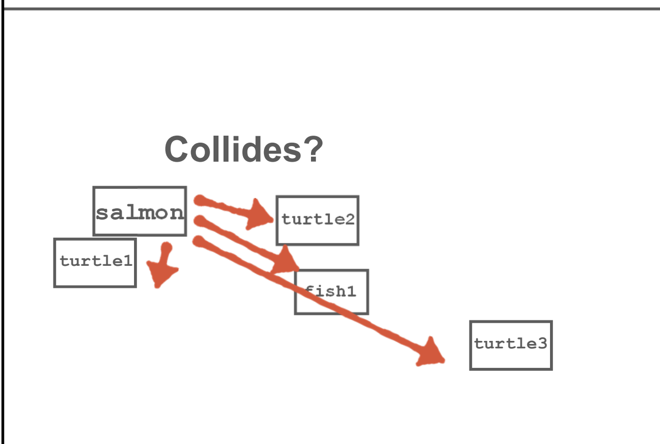


(0, 0)

21

Game State Update

bool World::update(float elapsed_ms)

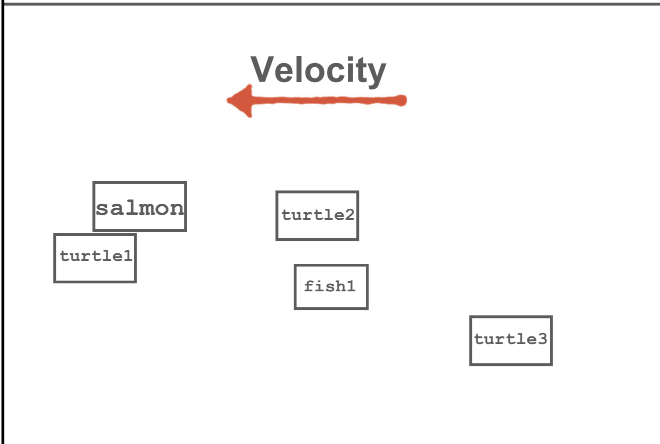


- Collision detection
  - Does salmon collides with turtle?
    —> Game over
    (Play audio and animation)
  - Does salmon collides with fish?
    —> Point gained

22

Game State Update

bool World::update(float elapsed_ms)

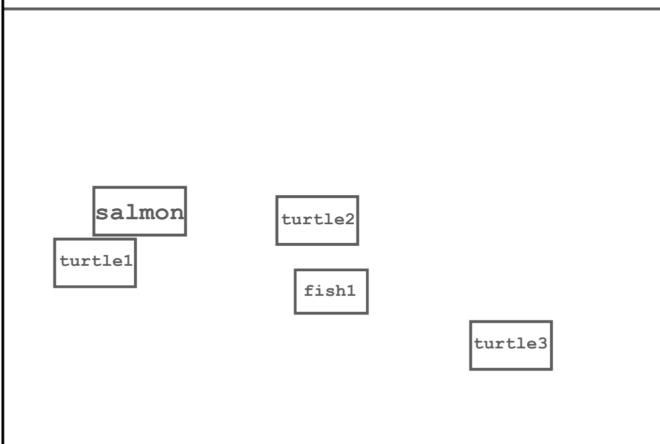


- Collision detection
- Update positions
  - Turtle and fish can have updated velocities
    ```
    -> turtle.update;
    fish.update
    ```
  - Update salmon position
    You need to move salmon.

23

Game State Update

bool World::update(float elapsed_ms)



- Collision detection
- Update positions
  - Turtle and fish can have updated velocities
  - Update salmon position

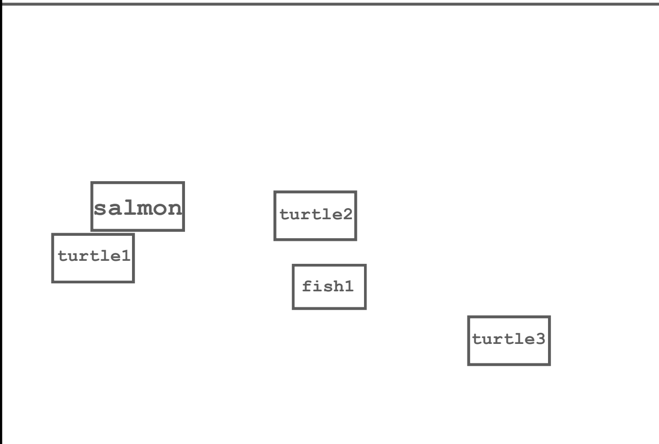
```
void World::on_key(GLFWwindow*, int
key, int, int action, int mod)
void Salmon::update(float ms)
```

Update salmon position using these two functions (hint: s = vt)

24

Game State Update

bool World::update(float elapsed_ms)

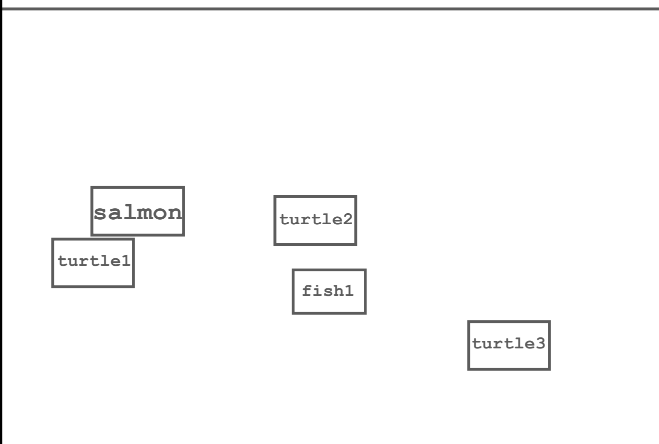


- Collision detection
- Update positions
- Remove fish and turtles that are outside

25

Game State Update

bool World::update(float elapsed_ms)

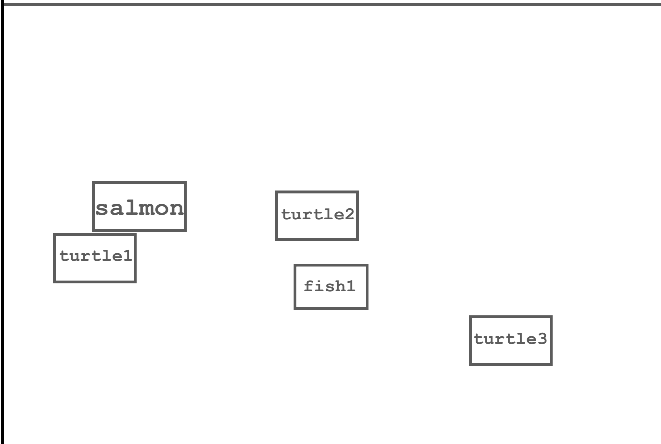


- Collision detection
- Update positions
- Remove fish and turtles that are outside
- Spawn fish and turtles

26

Game State Update

bool World::update(float elapsed_ms)



- Collision detection
- Update positions
- Remove fish and turtles that are outside
- Spawn fish and turtles
- If the game-over animation is over, restart the game

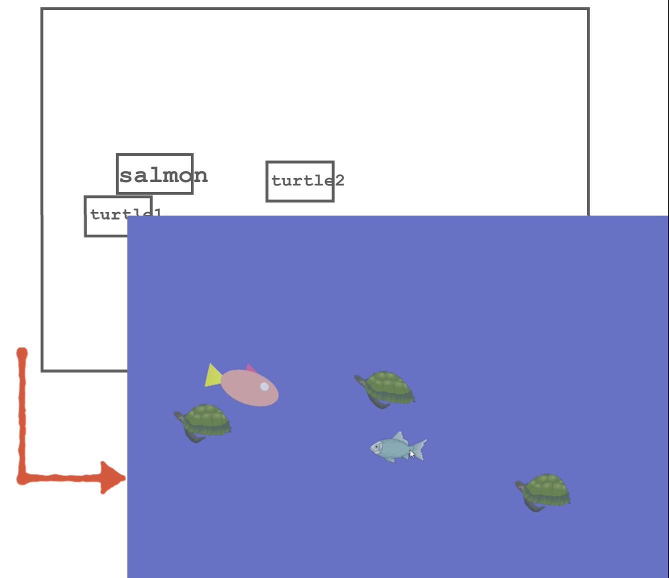
27

Mainloop

a1.cpp

```
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
        }
        ...
}
```

(2.1 Event processing and 2.2 Game state update are struck through; 2.3 Rendering a frame is highlighted.)



28

World: Draw

void World::draw() —> world.cpp:L268

{

2.3.1 Set window and depth range, clear the color and depth buffers

(Standard process before the actual drawing)

```
const float clear_color[3] = { 0.3f, 0.3f, 0.8f };
```

—> The background color in RGB format.

}

29

World: Draw

void World::draw() —> world.cpp:L268

{

2.3.1 Set window and depth range, clear the color and depth buffers

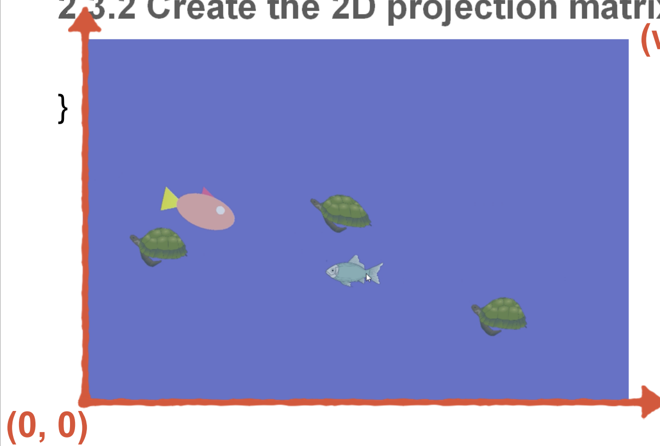
2.3.2 Create the 2D projection matrix

}

30

World: Draw

void World::draw() —> world.cpp:L268
{
…
2.3.2 Create the 2D projection matrix

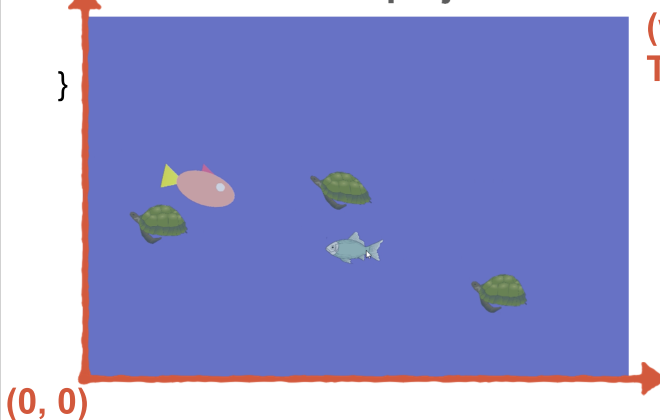


(width, height)

31

World: Draw

void World::draw() —> world.cpp:L268
{
…
2.3.2 Create the 2D projection matrix



(width, height)
This size varies!

32

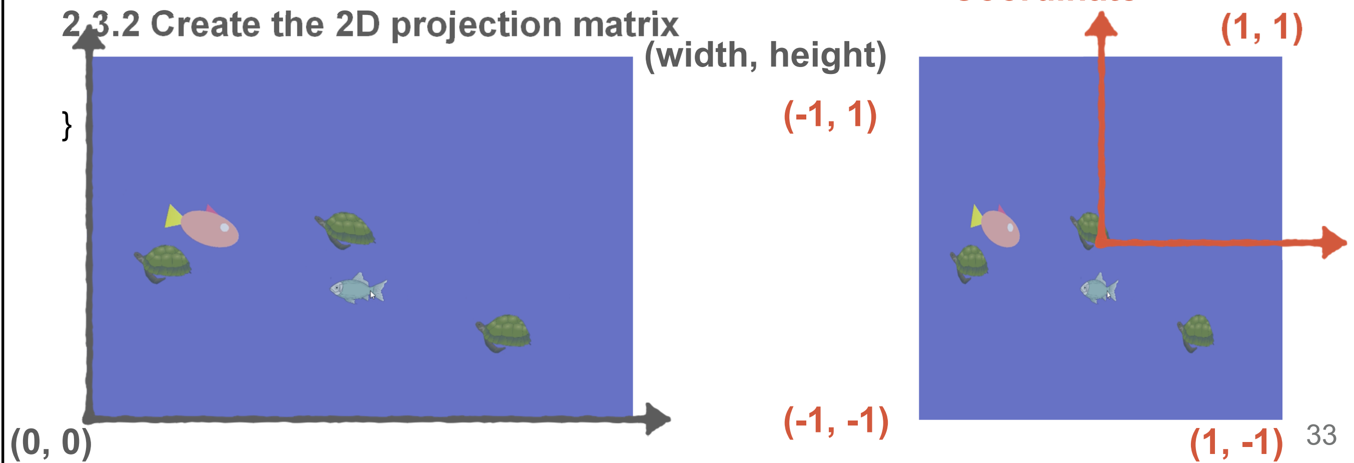
World: Draw

void World::draw() —> world.cpp:L268
{

...

2.3.2 Create the 2D projection matrix

}



33

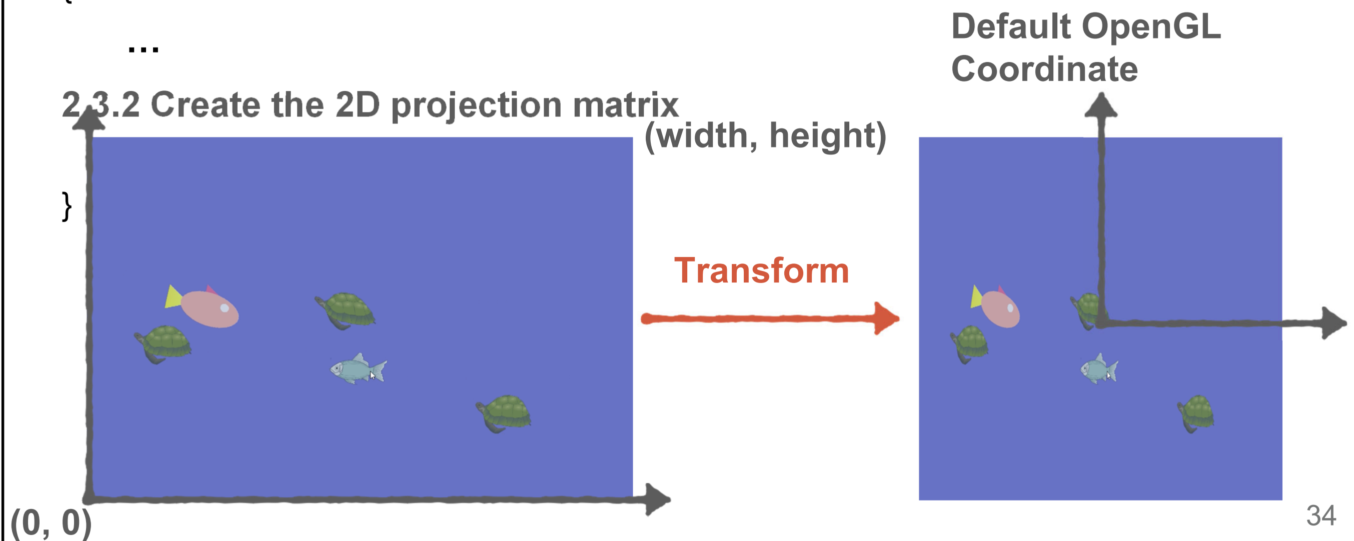
World: Draw

void World::draw() —> world.cpp:L268
{

...

2.3.2 Create the 2D projection matrix

}



34

World: Draw

```
void World::draw() —> world.cpp:L268
{
      ...
2.3.2 Create the 2D projection matrix
      float sx = 2.f / (right - left);
      float sy = 2.f / (top - bottom);
      float tx = -(right + left) / (right - left);
      float ty = -(top + bottom) / (top - bottom);
      mat3 projection_2D{ { sx, 0.f, 0.f },
                          { 0.f, sy, 0.f },
                          { tx, ty, 1.f } };
}
```

35

World: Draw

```
void World::draw() —> world.cpp:L268
{
      ...
2.3.2 Create the 2D projection matrix
}
```

In vertex shaders:

```
vec3 pos = projection * transform * vec3(in_position.xy, 1.0);
```

36

World: Draw

void World::draw() —> world.cpp:L268
{

2.3.1 Set window and depth range, clear the color and depth buffers

2.3.2 Create the 2D projection matrix

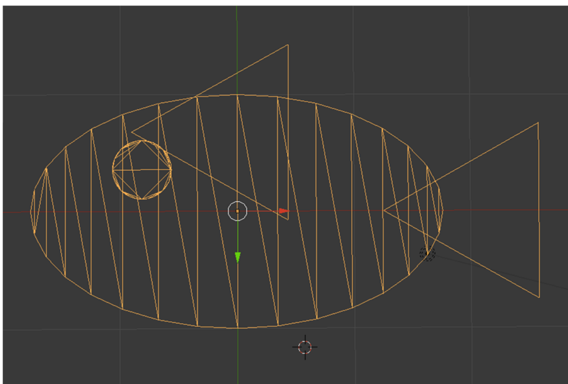
2.3.3 Draw individual entities (salmon, turtles, fish)

}

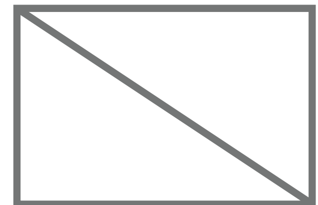
37

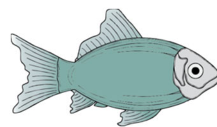
Salmon, Turtle, Fish: Appearances

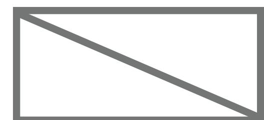
Salmon: 2D geometry



Turtle&Fish: Sprite
2D texture paint on simple geometry







38

Salmon, Turtle, Fish: Life Cycle

a1.cpp

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

1. world.init(windows_size) calls

   salmon.init;
   turtle.init;
   fish.init;

2. Mainloop:

   while (!world.is_over()) {…}

3. world.destroy() calls

   salmon.destroy;
   turtle.destroy;
   fish.destroy;

}

39

Salmon, Turtle, Fish: Life Cycle

Initializations

- Salmon: Load mesh; Turtle&Fish: Load texture, create the simple rectangles

40

Salmon, Turtle, Fish: Life Cycle

Initializations

- Salmon: Load mesh; Turtle&Fish: Load texture, create the simple rectangles
- Create VAOs and VBOs; Fill the buffers with vertex positions and indices

41

Salmon, Turtle, Fish: Life Cycle

Initializations

- Salmon: Load mesh; Turtle&Fish: Load texture, create the simple rectangles
- Create VAOs and VBOs; Fill the buffers with vertex positions and indices
- Load and create shaders

42

Salmon, Turtle, Fish: Life Cycle

Initializations

- Salmon: Load mesh; Turtle&Fish: Load texture, create the simple rectangles
- **Create VAOs and VBOs**; Fill the buffers with vertex positions and indices
- Load and **create shaders**

43

Salmon, Turtle, Fish: Life Cycle

Initializations

- Salmon: Load mesh; Turtle&Fish: Load texture, create the simple rectangles
- **Create VAOs and VBOs**; Fill the buffers with vertex positions and indices
- Load and **create shaders**

Remember to delete them in the destroy()!

44

Salmon, Turtle, Fish: Draw

```
void Salmon/Turtle/Fish::draw(const mat3& projection)
```

1. Specify OpenGL capabilities

```
glEnable(GL_BLEND); glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glDisable(GL_DEPTH_TEST);
```

45

Salmon, Turtle, Fish: Draw

```
void Salmon/Turtle/Fish::draw(const mat3& projection)
```

1. Specify OpenGL capabilities

2. Use the shader

```
glUseProgram(effect.program);
```

46

Salmon, Turtle, Fish: Draw

```
void Salmon/Turtle/Fish::draw(const mat3& projection)
```

1. Specify OpenGL capabilities

2. Use the shader

3. Compute parameters for the shader input (in our assignment the transform matrix)

In vertex shaders:

```
vec3 pos = projection * transform * vec3(in_position.xy, 1.0);
```

You need to compute it in:

```
void Salmon::draw(const mat3& projection)
```

47

Salmon, Turtle, Fish: Draw

```
void Salmon/Turtle/Fish::draw(const mat3& projection)
```

4. Send the correct data to shaders

    4.1 Look up the attribute location in a shader

In Salmon::draw

```
GLint in_position_loc = glGetAttribLocation(effect.program, “in_position");
GLint color_uloc = glGetUniformLocation(effect.program, “fcolor”);
```

48

Salmon, Turtle, Fish: Draw

```
void Salmon/Turtle/Fish::draw(const mat3& projection)
```

4. Send the correct data to shaders

4.1 Look up the attribute location in a shader

In Salmon::draw

```
GLint in_position_loc = glGetAttribLocation(effect.program, "in_position");
GLint color_uloc = glGetUniformLocation(effect.program, "fcolor");
```

In Salmon's vertex shader:

```
in vec3 in_position;
```

In Salmon's fragment shader:

```
uniform vec3 fcolor;
```

49

Salmon, Turtle, Fish: Draw

```
void Salmon/Turtle/Fish::draw(const mat3& projection)
```

4. Send the correct data to shaders

4.1 Look up the attribute location in a shader

In Salmon::draw

```
GLint in_position_loc = glGetAttribLocation(effect.program, "in_position");
GLint color_uloc = glGetUniformLocation(effect.program, "fcolor");
```

In Salmon's vertex shader:

```
in vec3 in_position;
```

In Salmon's fragment shader:

```
uniform vec3 fcolor;
```

Be careful about the uniform (aka global parameters).

50

Salmon, Turtle, Fish: Draw

```
void Salmon/Turtle/Fish::draw(const mat3& projection)
```

4. Send the correct data to shaders

    4.1 Look up the attribute location in a shader

    4.2 Send the data

In Salmon::draw, bind VAO and VBOs to correct locations:

```
glBindVertexArray(mesh.vao);
glBindBuffer(GL_ARRAY_BUFFER, mesh.vbo);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, mesh.ibo);
glEnableVertexAttribArray(in_position_loc);
glEnableVertexAttribArray(in_color_loc);
glVertexAttribPointer(in_position_loc, 3, GL_FLOAT, GL_FALSE,
    sizeof(Vertex), (void*)0);
glVertexAttribPointer(in_color_loc, 3, GL_FLOAT, GL_FALSE,
    sizeof(Vertex), (void*)sizeof(vec3));
```

51

Salmon, Turtle, Fish: Draw

```
void Salmon/Turtle/Fish::draw(const mat3& projection)
```

4. Send the correct data to shaders

    4.1 Look up the attribute location in a shader

    4.2 Send the data

In Salmon::draw, send the uniform attributes:

```
float color[] = { 1.f, 1.f, 1.f };
glUniform3fv(color_uloc, 1, color);
```

You need to set the correct current light-up state for this:

```
int light_up = 0;
glUniform1iv(light_up_uloc, 1, &light_up);
```

52

Salmon, Turtle, Fish: Draw

```
void Salmon/Turtle/Fish::draw(const mat3& projection)
```

1. Specify OpenGL capabilities
2. Use the shader
3. Compute parameters for the shader input (in our assignment the transform matrix)
4. Send the correct data to shaders
5. Finally, draw:

```
glDrawElements(GL_TRIANGLES,(GLsizei)m_num_indices, GL_UNSIGNED_SHORT, nullptr);
```

53

World: Draw

void World::draw() —> world.cpp:L268

{

2.3.1 Set window and depth range, clear the color and depth buffers

2.3.2 Create the 2D projection matrix

2.3.3 Draw individual entities (salmon, turtles, fish)

2.3.4 Swap the frame buffer (actually present the new frame to the window)

```
glfwSwapBuffers(m_window);
```

}

54

World: Draw

void World::draw() —> world.cpp:L268
{

2.3.1 Set window and depth range, clear the color and depth buffers

2.3.2 Create the 2D projection matrix

2.3.3 Draw individual entities (salmon, turtles, fish)

2.3.4 ~~Swap the frame buffer (actually present the new frame to the window)~~

—> We actually render using two passes.

}

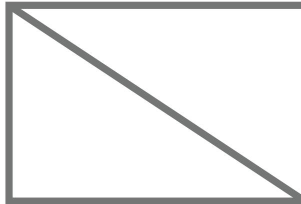
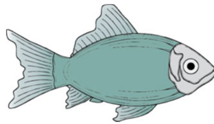
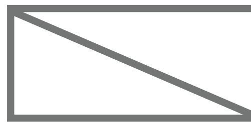
55

Why Two Passes?

- We want to add image-based post processing for the underwater effects:
  - Distortion
  - Color shifting

56

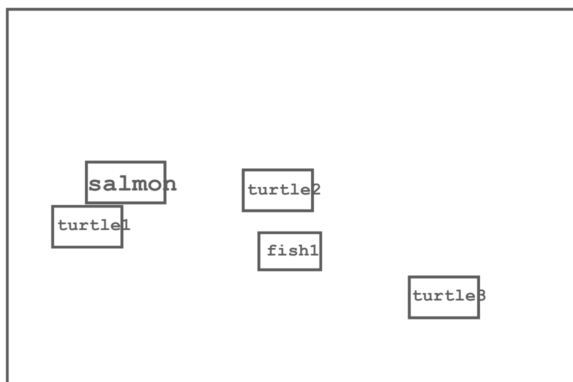
Recall Sprites…

57

Two Passes

- First pass

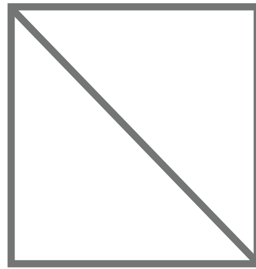


Render to Texture

58

Two Passes

- Second pass



In fragment shader:
- Distortion
- Color shift

A "sprite" covering the entire window

Final result

59

Two Passes

- I'm not going to talk about two-pass rendering in details
- If you are interested:
  - Check out water.hpp, water.cpp and World::draw
  - A detailed tutorial here:

http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-14-render-to-texture/

60

Water Fragment Shader

- In shader/water.fs.glsl, you need to implement:

vec2 distort(vec2 uv)

and

vec4 color_shift(vec4 in_color)

Water Fragment Shader

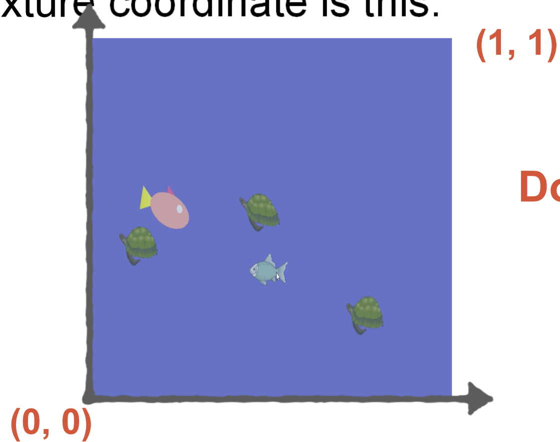
vec2 distort(vec2 uv)

Hint: use sin/cos, the uniform: time.

Water Fragment Shader

vec2 distort(vec2 uv)

Hint: use sin/cos, the uniform: time.
The texture coordinate is this:



Don't scale your sin/cos too muc

63

Water Fragment Shader

vec4 color_shift(vec4 in_color)

Colors here are in the RGBA format:
(R, G, B, A)

Each one of them is a float in the range [0, 1].

64

Water Fragment Shader

vec4 color_shift(vec4 in_color)

Colors here are in the RGBA format:
(R, G, B, A)

Each one of them is a float in the range [0, 1].

A is the alpha channel controlling the transparency.
A should always be 1 in this assignment.

65

Water Fragment Shader

vec4 color_shift(vec4 in_color)

Hint: To color shift to blue, just increase blue and decrease red and green.
(You don't need to match the solution exactly)

66

QUESTIONS?

TA: Chenxi
Reachable via Piazza and office hours

67