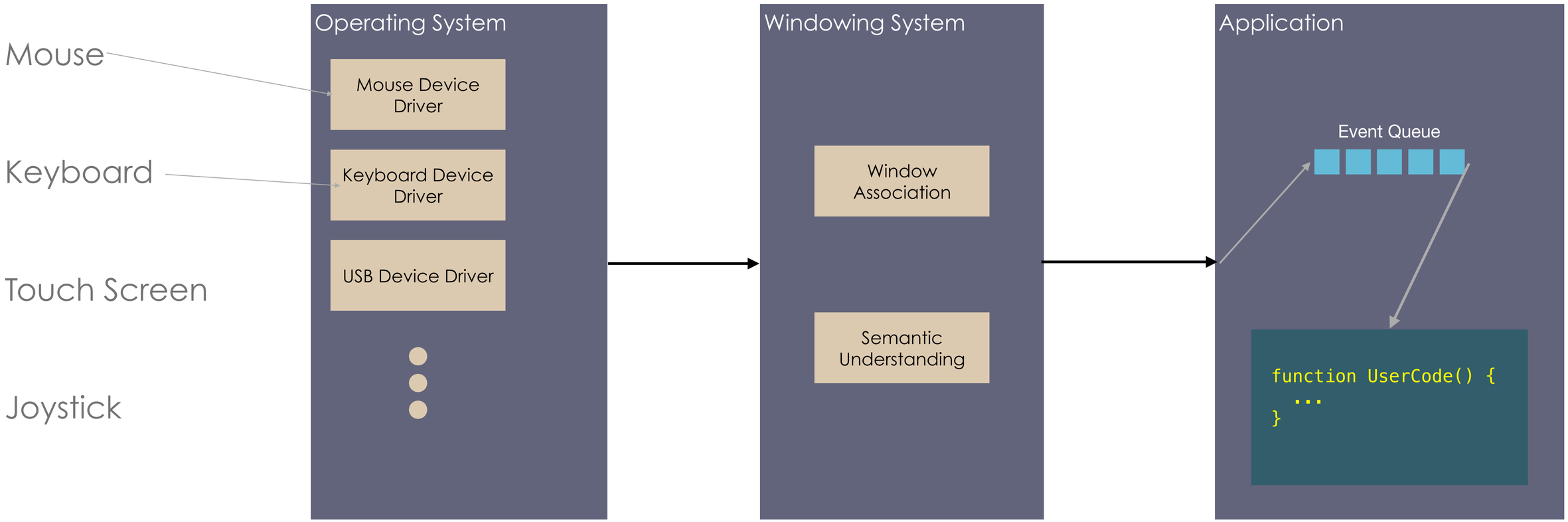
Input

CS 385 - Class 14
10 March 2022

Processing Events

Event Pipeline



The Main Loop

- Goes by a lot of names
  - Player Loop
  - Event Processing Loop
- For each frame
  - process all events in the queue
  - render using updated state
- This approach tends not to scale when an application has numerous UI elements
  - events are often relative to an element

```
while (!timeToExit) {
  while (getNextEvent(event)) {
    switch(event.type) {
      case MouseMove:
        // Do mouse things
        break;

      case MouseButton:
        switch(event.button) {
          case LeftMouseButton:
            // Do button press things
            break;
          ...
      } // MouseButton event.button switch
    } // event.type switch
  }

  render();
}
```

Asynchronous Approach

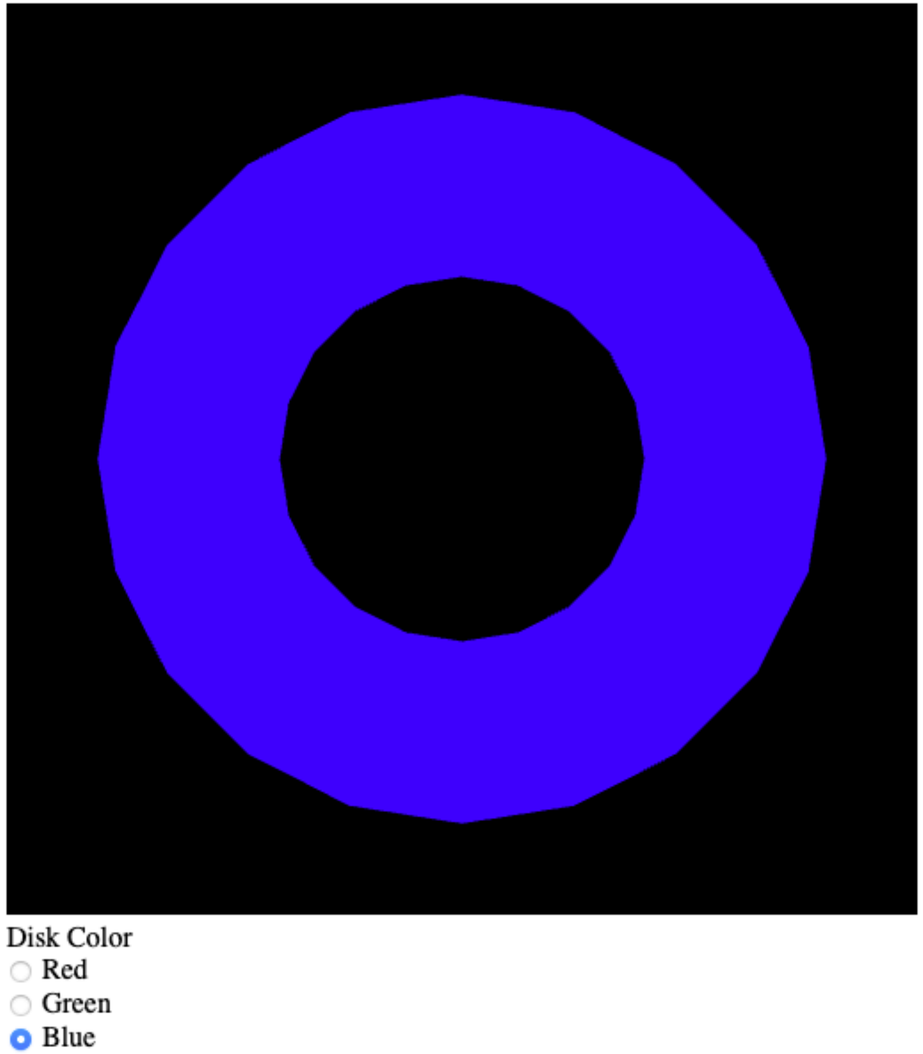
- Register a *callback function* to a particular element and event type
- When the "system" sees said event routed to the specific element, call the registered function
- JavaScript calls these callbacks *event listeners*

```
window.onload = init;


canvas.addEventListener("mousemove",
  function (event) {
    var x = event.clientX;
    var y = event.clientY;
    ... // update motion variables
  });


<form onchange="myEventProcessor()">
  ...
</form>
```

User Interfaces and Events

Buttons, Sliders, and Menus, oh my!

- Every windowing system provides some kind of *user interface* toolkit
  - *UI elements*
- Specific graphical representations for specific operations
  - Button: press and release
  - Slider: linear selection of values
  - etc.



Processing User Input

- Event listeners are registered directly with the UI element
- Each UI element has a set of supported actions that can have listeners
-

```html
<form onchange="setColor(event)">
    Disk Color
    <div>
        <input type="radio" name="Color" value="vec4(1,0,0,1)"><label>Red</label><br>
        <input type="radio" name="Color" value="vec4(0,1,0,1)"><label>Green</label><br>
        <input type="radio" name="Color" value="vec4(0,0,1,1)"><label>Blue</label><br>
    </div>
</form>
```

```javascript
function setColor(event) {
  disk.color = eval(event.target.value);

  requestAnimationFrame(render);
}
```

Not Quite Magic

- A little JavaScript trick
- Set UI element's `value` to code you'd like to use
  - a WebGL-suitable color value in our case
- `eval()` the returned string to convert it into executable JS

```
<form onchange="setColor(event)">
    Disk Color
    <div>
        <input type="radio" name="Color" value="vec4(1,0,0,1)"><label>Red</label><br>
        <input type="radio" name="Color" value="vec4(0,1,0,1)"><label>Green</label><br>
        <input type="radio" name="Color" value="vec4(0,0,1,1)"><label>Blue</label><br>
    </div>
</form>
```

```
function setColor(event) {
  disk.color = eval(event.target.value);

  render();
}
```

Basic Events

Keyboard Input

- Use key presses (and releases) for input
- Events received on the HTML *window*
  - our canvas is a subarea, so uses its parent for input
- `KeyboardEvent` passed to function
- The returned event contains:
  - `event.key` - return character
  - `event.keyCode` - ASCII character code
- Multiple event types
  - `keydown`
  - `keypress`
  - `keyup`

```
window.onkeydown = function (event) {
  switch(event.key) {
    case 'w': ... ; break;
    ...
  }
}
```

Mouse Input

- Mouse button presses and releases
- Events received on the HTML *element*
  - this works on our canvas
- `MouseEvent` passed to function
- Multiple event types
  - `mousedown`
  - `mouseup`
  - `mousemove`

```
canvas.onmousemove = function (event) {
    var x = event.clientX;
    var y = event.clientY;
    ...
  }
}
```

Mouse Input

- Only receive mouse movement when a mouse button is pressed down
- Register mousemove event listener when a mousedown event is seen
- Remove mousemove listener when a mouseup event happens

```
var startX;
var startY;

function mousemove(event) {
  var x = event.clientX;
  var y = event.clientY;
  var dx = x - startX;
  var dy = y - startY;
  ...
}

canvas.onmousedown = function (event) {
  var startX = event.clientX;
  var startY = event.clientY;

  canvas.addEventListener("mousemove", mousemove);
  }
};

canvas.onmouseup = function (event) {
  canvas.removeEventListener("mousemove", mousemove);
};
```

Motion & Trackballs

Math Time!