CS 418: Interactive Computer Graphics

(Very Simple) Event Driven Programming with JavaScript

Eric Shaffer

The Vertex Transformation Pipeline

- In the vertex shader we can process vertex positions:

```
uniform mat4 uMMatrix;  //Model matrix
uniform mat4 uVMatrix;   //View matrix
uniform mat4 uPMatrix;   //Perspective matrix
varying vec4 vColor;
void main(void) {
            gl_Position= uPMatrix*uVMatrix*uMMatrix*vec4(aVertexPosition, 1.0);
            vColor = aVertexColor;
}
```

- uMMatrix: Model transform from object coordinates to world
- uVMatrix: Transforms world to view coordinates
  - Usually combined: uMVMatrix= uVMatrix*uMMatrix
- uPMatrix: Transforms view coordinates to clip space

View Transformation

- Applied to all objects in the world uniformly
- Allows you to set a position for the eye in the world
- Applied after modeling transformations
  - So it should be the first transformation applied to the modelview matrix

- We will see a simple way to interactively change the eyepoint
  - This is not necessarily the best way to control a camera…

HTML Events

- An HTML event can be
  - something the browser does
  - something a user does
- Examples
  - an HTML web page has finished loading
  - an HTML input field was changed
  - an HTML button was clicked
  - user presses a keyboard key
  - user clicks a mouse button.

JavaScript lets you execute code when events are detected

DOM Level 2 Event Handling

- Different more sophisticated event handling
- Use addEventListener

```
document.addEventListener('keydown',
                                 handleKeyDown, false);
```

- Propagates events down and up DOM hierarchy
  - Multiple handlers can be invoked
- Probably not necessary for simple applications

jQuery Event Handling

- The jQuery library has it's own event handling capabilities
  - Useful…it was more cross-platform than DOM Event Handling
- However, most modern browsers handle events the same
  - So DOM event handling code you write should run everywhere
- You can use jQuery if you wish to

DOM Level 0 Event Handling

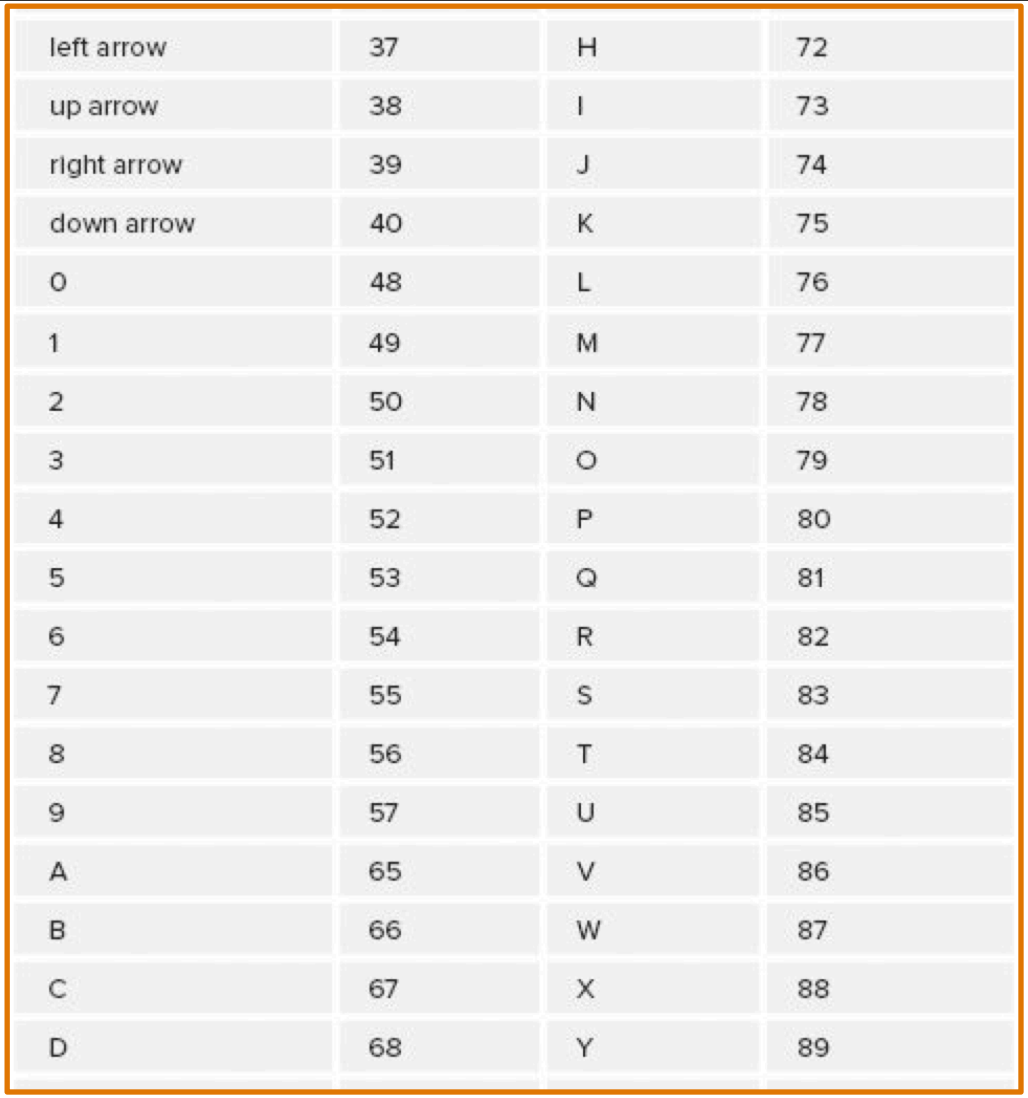
```
function startup() {
canvas =
document.getElementById
                ("myGLCanvas");
document.onkeydown =
handleKeyDown;
document.onkeyup =
handleKeyUp;
}
```

- Legacy event handling
- Supported in virtually all browsers
- Simple…usually sufficient

- Specify event handlers
  aka JavaScript functions
  - In HTML
  - or using DOM document in JavaScript…this is better as you can change event handlers dynamically

- DOM Level 0 event handling naming convention is ***onX*** where X is the event

Key Events



left arrow	37	H	72
up arrow	38	I	73
right arrow	39	J	74
down arrow	40	K	75
0	48	L	76
1	49	M	77
2	50	N	78
3	51	O	79
4	52	P	80
5	53	Q	81
6	54	R	82
7	55	S	83
8	56	T	84
9	57	U	85
A	65	V	86
B	66	W	87
C	67	X	88
D	68	Y	89

`keydown`
Physical key is pressed down

`keypress`
A character has been entered

`keyup`
Physical key has popped back up

event.keyCode is a numeric code that specifies which physical key was involved in the event

codes can vary by browser, so test with multiple browser for production work

Key Events

```
var currentlyPressedKeys = {};

function handleKeyDown(event) {
    currentlyPressedKeys[event.keyCode] = true;
}


function handleKeyUp(event) {
     currentlyPressedKeys[event.keyCode] = false;
}
```

Can use an associative array to collect multiple keydown and keyup events that occur between frames

Changing the Eyepoint

```
function handleKeys() {
	if (currentlyPressedKeys[37]  | |
	   currentlyPressedKeys[65]) {
	   // Left cursor key or A
	   eyePt[0]-= 0.2;
	} else if (currentlyPressedKeys[39]  | |
	           currentlyPressedKeys[68]) {
	   // Right cursor key or D
	   eyePt[0]+= 0.2;
	}
}
```

Call handleKeys once per frame

Set Up a View

```
var eyePt = vec3.fromValues(0.0,0.0,150.0);
var viewDir = vec3.fromValues(0.0,0.0,-1.0);
var up = vec3.fromValues(0.0,1.0,0.0);
var viewPt = vec3.fromValues(0.0,0.0,0.0);
```

Set Up a View

```
function draw() {
  // For a 2D game or visualization, ortho would be appropriate
  // We'll use perspective in anticipation of moving to 3D

  mat4.perspective(pMatrix,degToRad(45),
         gl.viewportWidth / gl.viewportHeight, 0.1, 200.0);

  // We want to look down -z, so create a lookat point in that direction
  vec3.add(viewPt, eyePt, viewDir);

  // Then generate the lookat matrix and initialize the MV matrix to that view
  mat4.lookAt(mvMatrix,eyePt,viewPt,up);
```

- We use glMatrix to set up a perspective view volume
- We create a view transformation matrix
  - and initialize the ModelView matrix to the view transformation

Questions

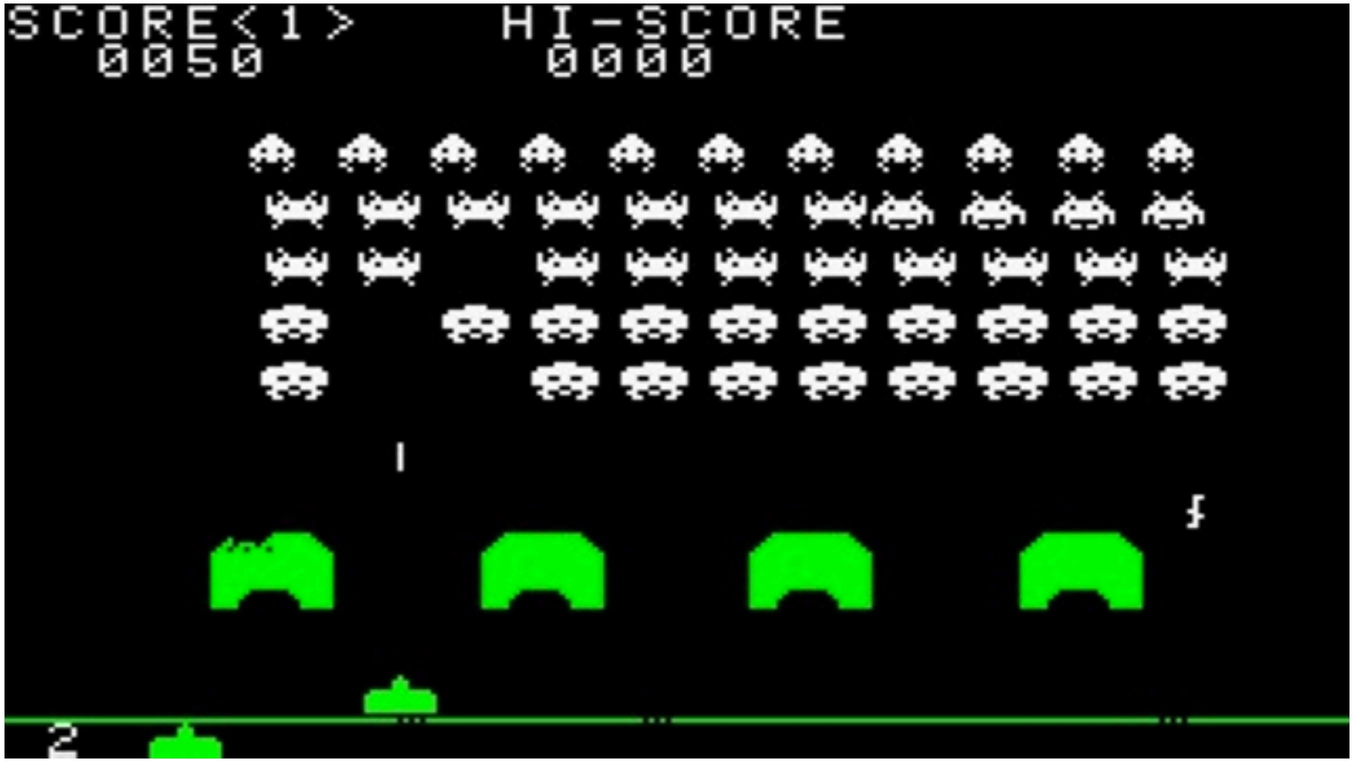
- When will the view transformation be applied to a triangle
  - Before or after modeling transformations?

- What if I want to move a single object interactively? What should be done?

- If you press the right arrow, objects appear to move left.

  Does this seem right?

You now know enough….



Developed by Tomohiro Nishikado and released in 1978