Week 3

9/18/17

Color in Computer Graphics

* How to get color?
* CMYK Model
  + Going to print
* HSV Model
  + Hue, Saturation, Value
  + Shader effect
* YUV Model
  + Used more in video processing
  + Luminance & Chrominance
* RGB Model
  + Most Commonly Used

Color in GL

* RGB and RGBA
* 0.0-1.0 floating point channels

Textures and Images

* STB\_image
* Opens into RAM
* #define STB\_IMAGE\_IMPLEMENTATION

Textures in OpenGL

* void glGenTextures (GLsizei numTextures, GLuint \*textures)
  + Generatess a new OpenGL texture ID
* void glBindTexture (GLenum target, GLuint texture)
  + Bind a texture to a texture target
* void glTexImage2D (GLenum target, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid \*pixels)
  + Sets the texture data of the specified texture target. Image format must be GL\_RGBA for RGBA imagees or GL\_RGB for RGB images

Texture Filtering

* Linear: Good for high resolution textures
* Nearest Neightbors: Good for pixel art
* void glTexParameteri (GLenum target, GLenum pname, GLint param)
  + Sets a texture parameter of the specified texture target
  + We must set the texture filtering parameters GL\_TEXTURE\_MIN\_FILTER and GL\_TEXTURE\_MAG\_FILTER before the texture can be used
  + Use GL\_LINEAR for linear filtering and GL\_NEAREST for nearest neighbor filtering

Texture Coordinates

* Texture coordinates are defined in 0-1 units called UV coordinates, not pixels
  + T(V) = y, going downward is positive
  + S(U) = x
  + Starts on top left
* void glVertexAttribPointer (Glindex, GLint size, GLenum type, GLboolean normalized, GLsizei stride, const GLVoid \*pointer)
  + Defines an array of vertex data

Drawing a Sprite

* Draw two triangles around the origin
* Need to use a shader program that supports textures

Blending

* glEnable(GL\_BLEND)

Basic gameplay programming

Movement

* Think in terms of time and frames
* Every frame ask the computer for the time
* Setup
  + Float lastFrameTicks = 0.0f
* In game loop
  + float ticks = (float)SDL\_GetTicks()/1000.0f
  + float elapsed = ticks – lastFrameTicks
  + lastFramTicks = ticks;
* elapsed is how many seconds elapsed since last frame. We will use this value to move everything in our game
* good practice to put f at the end to explicitly tell the computer it’s a float
* y\_position += elapsed \* distance\_to\_travel\_in\_one\_second
* a vector is like a number but it has a magnitude and direction
* a 2D vector is like a 2D coordinate, but has a magnitude and a direction
* Unit Vector
* Position += direction\_vector \* elapsed \* units\_a\_second
  + More accurate than the previous equation
* Position.x += cos(angle) \* elapsed \* units\_a\_second
* Position.y += sin(angle) \* elapsed \* units\_a\_second

Reading keyboard input

* Polling input vs input events
* Polling input = checking to see if a key is pressed. Useful for continuous player actions, such as movement, or checking modifier keys
* \*SDL\_GetKeyboardState(int \*numkeys)
  + Returns a pointer to an array of key states. A value of 1 means that the key is pressed and a value of 0 means that it isn’t. indexes into this array are obtained by using SDL scancode values. The pointer returned is a pointer to an internal SDL array. It will be valid for the whole lifetime of the application and should not be freed by the caller. We can passit a pointer to an int if we want to know the size of the array.
* SDL scancodes: All start with SDL\_SCANCODE
* To read input events, we use our event loop to see if the event has a type of SDL\_KEYDOWN or SDL\_KEYUP. We can then check the key that was pressed or released by checking the key member of the SDL event structure.

Collision Detection

* Circle- circle collision detection
  + Pythagorean theorem
  + If the distance between two circles is less than or equal to the sum of their radii, the circles are colliding
* Circle-point collision detection
  + If the distance between the point and the circle center is less than or equal to the radius.
* Box-Box collision Detection
  + Left side: rectangle.x – rectangle.width/2
  + Right side: rectangle.x + rectangle.width/2
  + Bottom/top: rectangle.y +- rectangle.height/2
* Box-Box conditions:
  + R1’s bottom is higher than R2’s top
  + R1’s top is lower than R2’s bottom
  + R1’s left is larger than R2’s right
  + R1’s right is smaller than R2’s left
  + If any of the above are true, then the two rectangles are not intersecting OR if none of the above are true they are intersecting
* Box-Point collision Detection:
  + Happens if point x is larger than box left and smaller than box right OR point y is larger than box bottom and smaller than box top

Assignment #1

* Check Slides