21M.385 Lecture Notes

Lecture 3

Anatomy of real-time app

- The main application loop - sometimes called game loop - is usually tied to the screen refresh rate, which is 60Hz. Game frame-rates are therefore 60Hz or sometimes 30Hz, giving 16.6ms (or 33.3ms) of time to draw a single frame.
- Every frame has 3 steps: Process input, Poll/Update animation state, Draw stuff on screen.
- Responding to user input (keyboard, mouse, gamepad, etc..) can happen in two ways:
  - Polling – manually query the state of input-device every time through the game loop.
  - Event driven – receive a callback - like `on_keydown()` – when something interesting happens
- Kivy framework (adapted for use in this class) has:
  - Polling – `on_update()` gets called every frame. Can query mouse position with `get_mouse_pos()`.
  - Events – `on_touch_down()`, `on_touch_move()`
  - Drawing happens automatically by Kivy. See object-based drawing below.
- Other examples of framework: Processing (Java), p5.js (Javascript), Unity (C#), Unreal (C++).

Read the Docs

- Read the Kivy docs: https://kivy.org/docs/gettingstarted/intro.html has a lot of good stuff. We will not use everything in this class (in particular, we are avoiding the Kv Design Language).
- And the Kivy API reference: https://kivy.org/docs/api-kivy.html

Example – mouse events and mouse polling

- We are familiar with keyboard events from before. You can also respond to mouse events:
- `on_touch_down()` 5
- `on_touch_up()` 5
- `on_touch_move()` 5
- And Polling mouse position with `get_mouse_pos()`
- Note 2D coordinate system - (0,0) is bottom-left.

Object-based drawing

- Unlike other frameworks, Kivy uses a list of *instruction objects* to render onto the screen. Drawing is done for you under the hood.
- This is not the same as immediate-mode drawing (like Processing), which uses *draw-commands*. In Processing, you must call "draw circle" every frame.
- In Kivy, to draw a circle, you instantiate a `Circle` object (well, `Ellipse`, actually) and add it to the `canvas` of the main window. The `canvas` is the list of instructions that Kivy will draw every frame.
- Two types of instructions:
  - Drawing Instructions (Ellipse, Rectangle, Line)
  - Context Instructions (Color, Translate, Rotate, Scale – described later)

Examples – a bunch of colored circles

- Add a circle to the canvas each time the mouse is clicked.
- Use `canvas.add(obj)` to add a drawable item to the canvas.
- To change the color, create a `Color` instruction and add it to the `canvas`
- Kivy then goes through the canvas instructions *in order*.
- Example of objects on the canvas stack:
  - Color(1,0,0)

- Ellipse(pos=(0,0), size=(50,50))
  - Color(0,1,0)
  - Ellipse(pos=(100,100), size=(30,30))
  - This will draw a red circle of diameter 50 and a blue circle of diameter 30.
- Note the *registration point* of an object (bottom left). Can add an offset so that circle is centered, or use helper class `CEllipse`.
- Keeping track of these objects allows us to animate or change their initial state:
  - modify an object's parameters. E.g., `self.color.rgb = (r, g, b)`
  - `canvas.remove(obj)` to remove an object from the instruction list. Only use `canvas.clear()` if no one else is using that canvas.

Kivy uses OpenGL

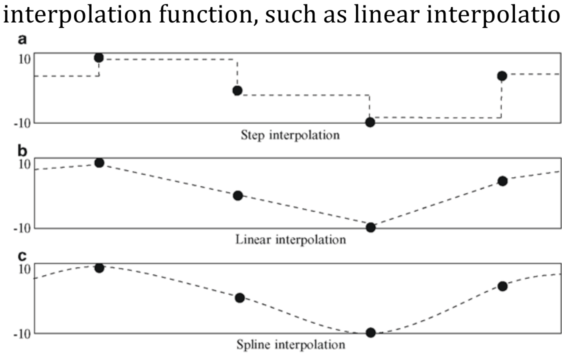
- Everything drawn boils down to triangles
  - Ellipse is really just a set of triangles. You can change # of segments to see this.
  - This is useful for 3D, which is what openGL is primarily about.
- Render State (color, location Matrix - Rotation, Translation, Scale)

Complex Instructions

- When making more complex drawing object, it is useful to encapsulate. A custom object can inherit from `InstructionGroup`. It behaves like a `canvas` (you can add instructions to it). But it also functions as an instruction itself. You can add this custom object into a different `canvas`.
- See the object `Bubble` in lecture3.py.

Key Frame Animation

- In key frame animation, a set of points are pre-defined over a time range. Each point is a time, value pair (called a key frame). To find a value at a specific time in between key frames, use an interpolation function, such as linear interpolation.



- See helper class `KFAnim` in gfxutil.py which defines values at points in time, and linearly interpolates between those values.
- `MainWidget4` shows animations of a circle's size and position.
- It is useful to have automatic *object lifetime management* based on updating an object's animation: `object.on_update()` returns `True` to keep going and `False` when object is done and should be removed. This uses the same philosophy as audio generators!

Dynamics / Physics Animation

- Dynamics-based animation systems are useful for animating motion. Positions are calculated via time-based function evaluation.
- In a physics-based system, Newtonian functions are calculated using numerical integration.
  - $v(t + \Delta t) = v(t) + a(t) \cdot \Delta t$
  - $x(t + \Delta t) = x(t) + v(t) \cdot \Delta t$
  - Collision is handled by reversing velocity and multiplying by a damping factor.
- `MainWidget5` shows a simple physics-based animation.

- Note that the basic animation framework is identical to the key frame system – `on_update()` is called and returns `False` when the object is done. In fact, this code has been encapsulated into a helper class called `AnimGroup`.

Reference Frames

- OpenGL supports reference frame instructions in addition to draw instructions. Each such instructions modifies the graphics context Transform matrix. These are:
  - Translate
  - Rotate
  - Scale
- Kivy has canvas instruction objects `Translate`, `Rotate`, and `Scale` that modify the graphics context accordingly.
- `MainWidget6` shows a simple example using Translate and Rotate.
- `PushMatrix` saves the current Transform Matrix. Later on, `PopMatrix` restores the Transform matrix to its previous value.
- `MainWidget7` draws a flower using these techniques.
- Any of these transforms can be referenced and be used later to animate portions of the reference frame tree.

More Graphics Examples

- A few more examples of primitives – Lines, Bezier lines, Rectangles, using textures, and color alpha. See *more_primitives.py*
- Dynamic Lines / Dots. Just lines and dots moving around. See *moving_dots.py*
- Mesh object – all OpenGL draw-objects are meshes. Meshes are collection of connected triangles that can form very complex 3D shapes. Textures (2D bitmaps) can be applied to Meshes. Mesh vertices can be animated. See *meshtest.py*
- Particle System – a large collection of textured squares (each a "particle") with dynamics-based animation, size animation, and color animation applied to all particles. Together they form some awesome looking effects. See *particle_paint.py*

Combining graphics and music

- Real-time graphics can reinforce the sound that we hear if there is a tight coupling (ie, a clear mapping) between sound and visuals.
- In the simplest case, you may have a one-to-one correspondence between notes and visual elements: one shape per note, with the duration of the note matching the duration of the shape.
- Graphics may have mismatched duration with music – visuals can remain visible longer than the sound to help remember events of the past. Visuals can disappear faster than the sound to highlight the appearance of new sounds.
- There are many graphical parameters to vary: shape, size, color (including hue, and brightness), texture, as well as different types of motion.
- There are many musical properties to illuminate: pitch, volume, timbre, note duration, rhythmic elements, tempo, chords, melodic lines, and abstract properties such as mood and energy.
- You can create mappings between musical properties and graphical parameters to highlight certain aspects of the music.
- One paper the addresses some of these ideas is: *Principles of Visual Design for Computer Music*, by Ge Wang.

Implementation Notes

- Callback functions are very useful in managing the code complexity. When an event is detected in an object (like a physics object), it can call a *callback function* to indicate that a particular event happened. That callback function (which is defined on a different object) can do non-graphical things like play a note.

21M.385 Interactive Music Systems
Fall 2016