(Modern) Animation
Quick Start

• Animation in computer graphics has a long and storied history, but before we go into that, we’ll jump right into how it’s generally done today, specifically in OpenGL (GLUT)

• Animation today is full-frame, double-buffered, and is actually not unlike traditional cel animation: present, in rapid order, a series of still images which the human brain interprets as fluid motion

• As a rule of thumb, we start perceiving “fluid motion” at around 30 frames per second

Drawing Full Frames

• You’ve now seen a single instance of OpenGL drawing, which generally goes like this:

  ◊ Clear the display
  ◊ Draw the objects

• The trick is, you don’t want the user to see the screen clear — this results in perceived flicker:
Enter Double Buffering

- The solution to the flickering issue is a technique called \textit{double buffering}: with double buffering, the system actually maintains two full screens — \textit{buffers} — at any given time; one is on display, and the other is invisible.

- Flicker-free animation is achieved by painting on the \textit{offscreen} buffer, then swapping the buffers.

- Graphics hardware is set up so that buffers can be swapped really quickly; drawing of the next frame then proceeds on the new offscreen buffer.

![Diagram of double buffering process](image)
Frame Rate and Real Time

• During double buffering, the frame rate is the frequency at which the buffers swap — conceptually equivalent to the “classic” frame rate of cel animation.

• However, remember that we’re on a computer system here — the frame rate is never absolutely the same, due to the amount of CPU time available, and the scheduling of other processes in the system.

• Thus, animation algorithms must be based on real time, not frame rate — otherwise, moving objects’ velocities will depend on CPU speed and scheduling.

• What you really want is for a more powerful system to render more frames per second than a less powerful system — but to still keep objects “moving” at the same “speed”.

• To do this, the classic mechanics equation applies:

  \[ \text{distance} = \text{rate} \times \text{time} \]

  1. Determine a rate/velocity/speed for moving objects in your program based on real time.

  2. When the CPU gives you time to display a frame, calculate how much time has passed since the last time you displayed a frame.

  3. Use \( d = rt \) to calculate how far to move the object in the new, upcoming frame.
GLUT Specifics

• Some specific GLUT calls/constants to remember when doing full-frame, real-time-based animation:

```c
glutInitDisplayMode(GLUT_DOUBLE | ... );
```

◊ Here, GLUT_DOUBLE sets up double buffering

```c
glutSwapBuffers();
```

◊ Call this when you’re done drawing; GLUT will make the buffer that you were using visible, and prepare the other buffer for the next time you draw a scene

• The function `int glutGet()` returns the current value of various GLUT variables. For animation, we want:

```c
int currentTime = glutGet(GLUT_ELAPSED_TIME);
```

◊ This will tell you the amount of time since some baseline value; what matters is how much time has passed since the last time you drew a frame

◊ Typically, you define a variable, say `lastTime`, and use `currentTime – lastTime` to determine the amount by which to move the objects in your program

```c
glutPostRedisplay();
```

◊ This asks GLUT to repaint your window — you almost never call the display function directly
Other Handy Animation Tricks

With this overall setup for animation, you can perform a few other useful things:

• *Calculate frame rate* — have a counter variable to increment whenever you generate a new frame; at regular intervals, use it to infer the overall frame rate

• *Cap frame rate* — generating a new frame (calculate new state + redisplay) can be expensive; you can give CPU time to other programs by doing nothing until the appropriate amount of time (e.g. at least 1/30th of a second) has passed