# Programming for Games

Alan Hazelden

# The game loop

- Real world is continuous
  - Computers: less so

```
int main (int argc, char** argv) {
    init();

    while (true) {
        update();
        draw();
    }

    finalise();
}
```

# Initialisation

- Create window
- Initialise world

```
int main (int argc, char** argv) {
    init();

    while (true) {
        update();
        draw();
    }

    finalise();
}
```

# Update

- Simulate a small period of time
  - Movement of game entities

```
int main (int argc, char** argv) {
    init();

    while (true) {
        update();
        draw();
    }

    finalise();
}
```

# Update

- Simulate a small period of time
  - Movement of game entities

```
void update () {
    for (int i = 0; i < entities.size(); i++) {
        entities[i].update();
    }

    // update game state
}

void Entity::update () {
    position += velocity;

    // check if we've collided with anything, etc.
}
```

# Draw

- Draw the new state of the world

```
int main (int argc, char** argv) {
    init();

    while (true) {
        update();
        draw();
    }

    finalise();
}
```

# Draw

- Draw the new state of the world

```
void draw () {
    for (int i = 0; i < entities.size(); i++) {
        entities[i].draw();
    }

    // draw anything else, e.g. UI
}

void Entity::draw () {
    sprite.draw(position.x, position.y);
}
```

# Framerate independence

- Runs differently on different computers
- Even on the same computer, speed can fluctuate

```
int main (int argc, char** argv) {
    init();

    while (true) {
        update();
        draw();
    }

    finalise();
}
```

# Framerate independence

- Time the execution of the loop

```
int main (int argc, char** argv) {
    init();

    int time = timer.getTime();

    while (true) {
        int elapsed = timer.getTime() - time;
        time += elapsed;

        update(elapsed);
        draw();
    }

    finalise();
}
```

# Framerate independence

- Simulate world for given length of time

```
void update (int t) {
    for (int i = 0; i < entities.size(); i++) {
        entities[i].update(t);
    }

    // update game state
}

void Entity::update (int t) {
    position += velocity * t;

    // check if we've collided with anything, etc.
}
```

# Framerate independence

- Problems
  - What if your computer is too slow?
    - Tunnelling
  - What if your computer is too fast?
    - Excess work
  - Inconsistent behaviour
    - Variable acceleration
    - Nondeterministic

# Framerate independence

- An alternative

```
int main (int argc, char** argv) {
    init();

    int time = timer.getTime();
    const int timestep = 16; // 16ms == 1/60s

    while (true) {
        update();
        draw();

        int elapsed = timer.getTime() - time;

        if (elapsed < timestep) {
            sleep(timestep - elapsed);
        }

        time = timer.getTime();
    }
```

# Framerate independence

- Problems
  - What if your computer is too slow?
    - Game runs slowly
  - What if your computer is too fast?
    - Hopefully nothing

# Framerate independence

- Where is the bottleneck?
  - In `update()`?
    - E.g. physics-intensive games
  - In `draw()`?
    - E.g. modern game with extra shininess
    - Potential solution: update more frequently than drawing

# Cameras

- Recall the `draw()` code
  - No way to view a different section of world

```
void draw () {
    for (int i = 0; i < entities.size(); i++) {
        entities[i].draw();
    }

    // draw anything else, e.g. UI
}

void Entity::draw () {
    sprite.draw(position.x, position.y);
}
```

# Cameras

- Simple solution: offset everything
  - Still can't zoom or rotate camera

```
void draw () {
    for (int i = 0; i < entities.size(); i++) {
        entities[i].draw(cx, cy);
    }

    // draw anything else, e.g. UI
}

void Entity::draw (float cx, float cy) {
    sprite.draw(position.x - cx, position.y - cy);
}
```

# Cameras

- Better: use matrices
  - (Java: AffineTransform class)

```
void draw () {
    glPushMatrix();
    glTranslate(-cx, -cy);
    for (int i = 0; i < entities.size(); i++) {
        entities[i].draw();
    }
    glPopMatrix();

    // draw anything else, e.g. UI
}

void Entity::draw () {
    sprite.draw(position.x, position.y);
}
```

# Questions?