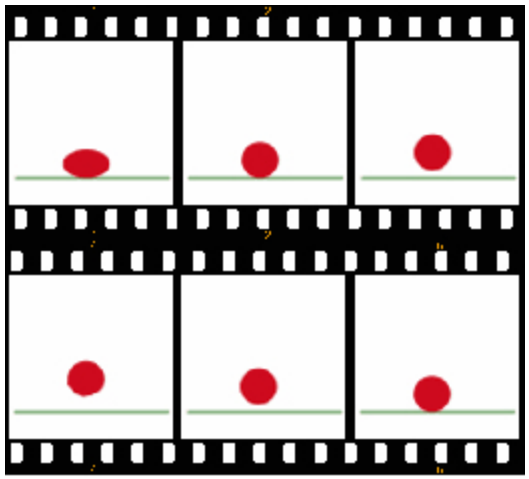
Animation

Thumrongsak Kosiyatrakul
tkosiyat@cs.pitt.edu

Animation

- **Animation** (Wiki) is a method in which pictures are manipulated to appear as moving images.



- In OpenGL, to change the location of an object, we need to **redraw**
  - The `glutPostRedisplay()` function
  - Send one or more transformation matrices before redrawing
- Thus, we need something that allows us to call the `glutPostRedisplahy()` multiple times

Calling the glutPostRedisplah()

- A way to call the glutPostRedisplay() multiple times is to use a type of loop:

```
for(i = 0; i < 50; i++)
{
    :
    glutPostRedisplay();
}
```

- Unfortunately, an OpenGL application is an **Event-Driven** program
  - A specific function (**Callback function**) will be called if an event occurs
    - mouse click
    - mouse move
    - keyboard
  - **Rule of Thumb**: A callback function should be executed as fast as possible
    - There may be other events waiting

Event-Driven with OpenGL

- If there is no event, the program is idling and waiting for an event
- In OpenGL, we call the `glutMainLoop()` function
  - Almost at the end of the `main()` function
  - Same effect as an infinite loop but it is looking for an event
  - If there is no event, simply wait
- We are going to use idle periods to redraw:
  1. Check the event queue:
     - If there is an event in the queue, process it and go back to step 1
     - If there is no event go to step 2
  2. Redraw the scene with new transformation matrices and go back to step 1
- This can be done by enabling idle callback

Enabling Idle Callback

- To enable idle callback, we use the `glutIdleFunc()` function:

```
void glutIdleFunc(void (*func)(void));
```

- Example:

```
:
void idle(void)
{
    :
}
:
int main(void)
{
    :
    glutIdleFunc(idle);
    :
}
```

- The `idle()` function will be call if there is no event

Disabling Idle Callback

- To disable the idle callback simply execute:

```
glutIdleFunc(NULL);
```

- A flag can also be used to bypass an animation process:

```
:
int isAnimating = 1;
:
void idle(void)
{
    if(isAnimating)
    {
        :
    }
}
:
```

Multiple Animations

- Recall Project 2, there are multiple kinds of animations:
  - Flying around the maze
  - Flying down to the entrance
  - Walk forward on cell
  - Turn left 90 degrees
  - Turn right 90 degrees
- It may help to have a state variable

```
typedef enum
{
    FLYING_AROUND = 0,
    FLYING_DOWN,
    WARK_FORWARD,
    TURN_LEFT,
    TURN_RIGHT,
} state;

state currentState = FLYING_AROUND;
:
```

Multiple Animations

- With the currentState variable, you can do the following:

```
void idle(void)
{
    if(isAnimating)
    {
        if(currentState == FLYING_AROUND) {
            :
        }
        else if(currentState == FLYING_DOWN) {
            :
        }
        else if(currentState == WALK_FORWARD) {
            :
        }
        else if(currentState == TURN_LEFT) {
            :
        }
        else if(currentState == TURN_RIGHT) {
            :
        }
    }
}
```

Speed of Animation

- On different machine, the `idle()` function is called at a different speed
- On the same machine, it may get call less frequent if the machine is busy
- To control the speed of animation, we generally use time
  - Record the start time t_0
  - The current transformation is based on the the different between the start time and the current time

$$\text{Transformation Matrices} = f(t - t_0)$$

  - Hard to do

Speed of Animation

- For this project, simply adjust the amount of changes between each idle called
  - Rotate an object 0.1 degree more vs 1 degree more each idle called
  - Move an object 1% more vs 5% more
- Or adjust the number of steps to perform a specific animation
  - Rotate this object for 90 degrees in 50 steps vs 100 steps
  - Move an object from here to there in 10 steps vs 20 steps
- These methods are easy to implement

Animation by A Number of Steps

- Each kind of animation will have its own number of steps to complete
- Examples
  - Fly around the maze in 100 steps
  - Fly down to the entrance in 50 steps
  - Walk forward on cell in 25 steps
  - Turn left 90 degrees in 20 steps
  - Turn right 90 degrees in 20 steps
- Note that speed of animation can be adjusted by changing the number of steps
- For this kind of implementation we need two global variables:

```
int current_step = 0;
int max_steps = 100;
```

Animation by A Number of Steps

- Every time the `idle()` function is called, increase `current_step` by 1
- Example (type of animations is omitted)

```
void idle(void)
{
    if(isAnimating)
    {
        current_step++;

        if(current_step == max_steps)
        {
            :
        }
        else
        {
            :
        }
    }
}
```

Changing Vector

- For each type animation it has a starting value(s) and a target value(s)
  - Rotation: Starting at 90 degrees and rotate to 180 degrees
  - Moving: Starting at (x, y, z) and move to (x', y', z')
  - Changing Camera Location and orientation:
    - **Eye**: from (x_e, y_e, z_e) to (x'_e, y'_e, z'_e)
    - **At**: from (x_a, y_a, z_a) to (x'_a, y'_a, z'_a)
    - **Up**: from (x_u, y_u, z_u) to (x'_u, y'_u, z'_u)
- Thus, each type of animation has its own changing vector(s) (1, 2, or more dimensions)
  - Rotation: $\mathbf{v} = 180 - 90 = 90$
  - Moving: $\mathbf{v} = (x', y', z') - (x, y, z) = (x' - x, y' - y, z' - z)$
  - Changing Camera Location:
    - $\mathbf{v}_{\text{eye}} = (x'_e, y'_e, z'_e) - (x_e, y_e, z_e) = (x'_e - x_e, y'_e - y_e, z'_e - z_e)$
    - $\mathbf{v}_{\text{at}} = (x'_a, y'_a, z'_a) - (x_a, y_a, z_a) = (x'_a - x_a, y'_a - y_a, z'_a - z_a)$
    - $\mathbf{v}_{\text{up}} = (x'_u, y'_u, z'_u) - (x_u, y_u, z_u) = (x'_u - x_u, y'_u - y_u, z'_u - z_u)$

Changing Vector

- With a changing vector $\mathbf{v}$, an intermediate point of changing from starting to target is

$$\text{current} = \alpha\mathbf{v} + \text{starting}$$

  where $0 \leq \alpha \leq 1$
- During an animation process, α should progressively change from 0.0 to 1.0.
  - Recall that `current_step` is progressively changed from 0 to `max_steps`
  - Obviously, α is based on the `current_step` and `max_step`.

$$\alpha = \frac{\texttt{current\_step}}{\texttt{max\_steps}}$$

Animation by A Number of Steps

- Thus we have something like the following (type of animation is omitted):

```
void idle(void)
{
    if(isAnimating) {
        current_step++;

        GLfloat alpha = (float) current_step / max_steps;

        if(current_step == max_steps) {
            isAnimation = 0;
            current_step = 0;
            max_step = 50;
        }
        else {
            current_something = (alpha * changing_vector) + starting;
            calculate new transformation(s)
        }

        glutPostRedisplay();
    }
}
```