Animation

CS134

Chris Pollett

Sep. 15, 2004

Introduction

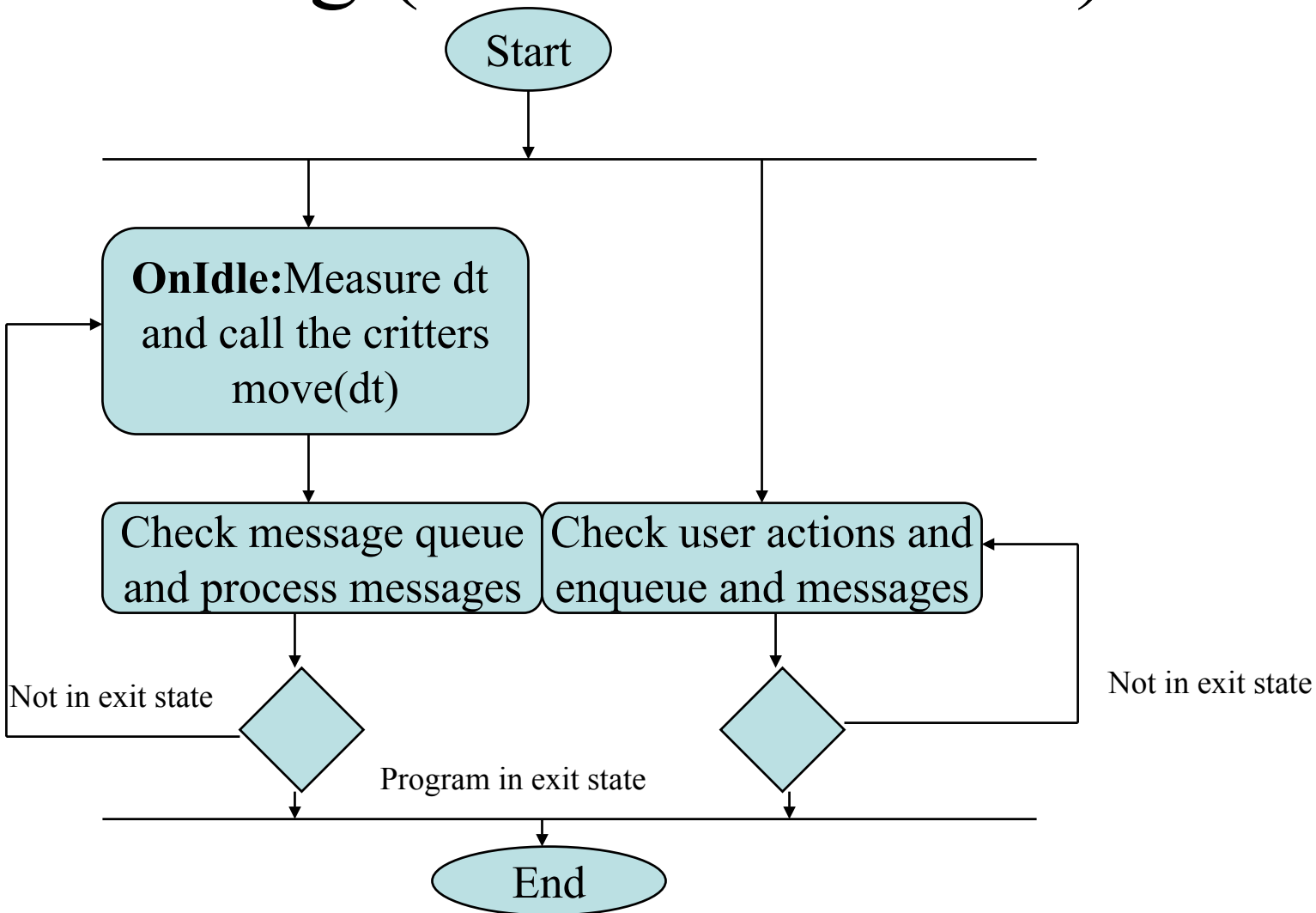
Interested in writing programs which continually update objects on the screen. We'll talk about:

- Creating endless animation loops
- Simulating speed in a processor independent fashion
- Cascading animation activities
- Updating different views of the animation

Creating endless animation loops

- In a typical game images on screen change even if you are not giving any input.
- Where in the program should this continual updating be done?
- How can one synchronize inputs with game updates?
- A UML diagram called an activity diagram is useful for describing how this is done in Pop.

Activity diagram of continuous drawing (like a flow chart)



More on Activity Diagrams

- In such a diagram diamonds indicate test points.
- Arrows show the flow of program control. Horizontal lines indicate forks and joins of parallel operations
- OnIdle will get called over and over and this is where we should put our animation code.
- Above can be viewed as an example of the Command pattern.

More on OnIdle

- We animate by overriding CWinApp::OnIdle.
- Want to make calls to the CDocument objects which will cascade down to the cGame objects.
- OnIdle is called at least once each time that it finishes processing its current messages
- OnIdle has a return type BOOL. true = keep calling OnIdle even if no messages.

First pass at Overriding OnIdle

```
BOOL CPopApp::OnIdle(LONG lCount)
{
    CWinApp::OnIdle(lCount);
    animateAllDocs(0.05);
    return TRUE;
}
```

Issues with First Pass

- 0.05 is supposed to indicate number of seconds between updates. However, would like to also be able to have this reflect how long it takes to update.
- To do this we'll use cPerformanceTimer which encapsulate the clock calls needed figure out much time has passed. Has a tick method. In addition:
  - Would like to be able to turn animation on and off.
  - Would like to minimize impact of code on other running processes.

Simulating speed in a processor independent fashion

- What value should be used for an animations dt where dt is our smallest timestep?
- If we update a point according to a rule like: new_pos = old_pos +dt*vel, want it to look the same on different speed hardware.
- So dt should be a time in second not processor cycles.

First pass at Overriding OnIdle

```
BOOL CPopApp::OnIdle(LONG lCount)
{
    CWinApp::OnIdle(lCount);
    double dt = _timer.tick();/*uses system clock to
       find out time elapsed */
    animateAllDocs(dt);
    return TRUE;
}
```

Size of dt

- Suppose on faster hardware get 50 updates a second; whereas on slower, hardware have 25 updates a second.
- In the first case, dt= .02 and in the second case dt =.04.
- So in first case, when do an update calculate new_pos =old_ pos +.02*vel and in the second case new_pos =old_pos+.04*vel.
- So motion is finer/smoother on faster hardware.

Comment on velocity

- By default critters in Pop have a default speed of 2.0. Then if window world size is 10 wide can expect it takes 5 seconds for an object to cross screen.
- Time to cross will not depend on hardware.

Measuring a Timestep

- What does tick() member function look like?

cPerformanceTimer
double _currenttime double _dt

cPerformanceTimer(){_currenttime =
getsystemtime();}
double getsystemtime();
double tick()
{
double systemtime = getsystemtime();
_dt = systemtime - _currenttime;
_currenttime = systemtime;
return _dt;
}

More on cPerformanceTimer

- getsystemtime is a private function of this class. Either uses clock() on old systems or timeGetTime() on newer systems. Latter is fairly precise as based on machine's processing speed.
- When you pause the game (for example because of requiring a dialog to be filled), you should call _timer.tick() to prevent jerkiness in motion.
- Beside dt, also have _maxdt and _mindt. The former is used to allow smooth motion on old machines (even if things must look slower). The latter is to prevent the graphics pipeline from being choked if processor speed is to fast.

Graphics Refresh Rate

- To figure out what _mindt should be call ::GetDeviceCaps(hdc, VREFRESH). Then take_mindt to be inverse of this.
- To guarantee _mindt has passed since the last refresh we can these have process spin, in tick(), in a while loop until greater than this amount of time has passed

Improving Animation Speed

- Speed depends on amount computation and graphics overhead required to put an object on the screen.
- If N critters might possible interact with any amongst themselves, could require O(N^2) time to compute the total interaction.
- Usually not doing this so graphics overhead will tend to dominate.
- Exact cost will depend on if using MFC or OpenGL.

Pixel Overhead

- How much time does it take to render the pixels in a frame?
- pixel overhead≈area of rectangle *colors per pixel * bus overhead.
- The number of pixels you are moving grows as the square of the edge dimension.
- On same hardware a 800x600 frame renders 4 times faster than a 1600x1200 one.

Number of Colors

- Some games take over the screen and run in single task mode.
- These games might then set the color depth as it likes in order to reduce pixel overhead.
- This is not done in Pop.
- Another way to speed things up is to increase the refresh rate on the graphics card, if possible.

Bus Overhead

- Time to move a pixel from one memory location to another.
- Generally, build up scene in an off-screen graphics area, then when ready move whole screen to frame buffer which graphics card displays.
- Bus overhead depends on graphics card. This moving can be avoid to some degree using page-flipping
- Used to be page-flipping only worked for whole screens.
- When possible OpenGL does page flipping for windowed apps

Cascading animation activities

What things are called by animateAllDocs(dt)?

- This function calls CPopDoc::stepDoc(dt) for each CPopDoc associated with game.
- This latter method calls cGame::step(dt) for the game inside the CPopDoc.
- step(dt) updates the positions and appearances of an array of critters in the game
- CPopDoc::stepDoc(dt) then calls CPopDoc::UpdateAllViews which in turn calls each cPopView's OnDraw
- OnDraw uses the CPopView's cGrpahics *_pGraphics member to draw on the screen view window.

stepDoc

```
void CPopDoc::stepDoc(real dt)
{
    _pgame->step(dt);
    cTimeHint timehint(dt);
    UpdateAllViews(NULL, 0, &timehint);
}
```

Updating different views of the animation

- UpdateAllViews(CView *pSender, int Lhint, CObject *pHint) generate calls to CPopView::OnUpdate(CView *pSender, int LHint, CObject *phint)
- This might just call Invalidate() to enqueue a message which will be handled by OnDraw.
- Could do calls like GetDocument()->UpdateAllViews(…) if wanted one view to be able to contact another.
- LHint 0 for normal updates, CPopDoc::VIEW_START_GAME for start of game