Interactive Graphics Systems

Pictures: two details of HCI from movie T*he Dark Knight (credits BLIND, Ltd.)*

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Animation

v1.0 20221115

Alexandre Valle | DEI | FEUP

Notes on animation (with sample code)

Animation
Animation based on pre-calculated increments
Animation based on effective elapsed time

1.Animation

Change of object parameters across a limited period of time.

Examples:

- The continuous rotation of a 3D object across a time interval
- The continuous displacement of a 3D object across a time interval

- *Visual Feedback* to highlight the selection of a 3D object
    - The periodic change of the color intensity (*glow*)

Animation based on pre-calculated increments

Considering P_{change} the total change of a property and considering the animation of the property encompasses n time instants, then

Between two consecutive time instants the change of the property is: P_{change} / n. Then the property will be affected by that amount:

P += P_{change} / n

Example: for and object displacement of 10 unit in the X axis over a period of 20 seconds, the change in displacement between two consecutive milliseconds is:

location.x += 10 / 20 * 1000

The previous assumes that, at millisecond zero, the location.x of the 3D object is set to its initial state.

Animation based on pre-calculated increments

Advantages: this type of animation is very simple and easy to apply if the time rate is uniform (evenly distributed across the time period) and allows the update of properties at a given frame rate.

Disadvantages: it assumes that the time between animation updates is constant. If this is not guaranteed (usually it is not guaranteed) the animation of the object will delay of speedup in a non-uniform fashion.

Animation based on effective-time

Assumptions:

- no control of exact time between property updates
- requirement to perform the entire animation of property *p* over a defined time amount:
    - Total duration of the animation: T_{total}
    - Total change in property: $P_{change} = P_{final} - P_{initial}$

Animation based on effective elapsed time

At the start of the animation:

- time $T_{initial} = t$ (current time)
- current property value $p = P_{initial}$

For each animation update:

- Get the current time, t, and calculate how much it has passed between the start of the animation and the current moment: $T_{span} = t - T_{initial}$
- If there is still time to update the animation (*e.g.* $T_{span} < T_{total}$), calculate the property value for the current time:

$$p = P_{initial} + P_{change} \times (T_{span} / T_{total})$$

Animation based on effective elapsed time

Important notes:

- T_{span} / T_{total} is the ratio of elapsed animation in [0..1]. When the ratio is equal to 1 it represents the end of the animation and a value beyond that means the animation should already be finished.

- P_{change} x (T_{span} / T_{total}) is the amount of property <u>change</u> corresponding to the ratio of the elapsed animation.

Animation based on effective elapsed time

Example: Given a 3D object, it is expected to perform the following animation over a T_{total} of 5 seconds (5000 milliseconds)

- $P1_{change}$ = (a rotation of) +90º = ($P1_{final} – P1_{initial}$) = 135º - 45º
- $P2_{change}$ = (a translation of) +10 units = ($P2_{final} – P2_{initial}$) = 17 - 7
- $P3_{change}$ = (a scale of) +20% = ($P3_{final} – P3_{initial}$) = 120% - 100%

Hence, for a given t where $t >= T_{initial}$

- $ratio = (t – T_{initial}) / T_{total}$

If *ratio <= 1.0*, then:

- $P1 = P1_{initial} + P1_{change} \times ratio$
- $P2 = P2_{initial} + P2_{change} \times ratio$
- $P3 = P3_{initial} + P3_{change} \times ratio$

else, *t* is beyond the animation time frame.

Animation based on effective elapsed time

Advantages: this type of animation is a bit more complex but is “well behaved”: non-periodic updates (that is, having delays) are accommodated because ratio is a measure considering the initial time and not time deltas.

Developing animation with WebCGF

```
class MyScene extends CGFscene {

    onGraphLoaded() {
        …
        // set the update call every 100ms
        // zero to disable it
        this.setUpdatePeriod(100);
        this.startTime = null;
    }

    // called if update period > 0
    // browser will try to comply with update period
    update(time) {
        if (this.sceneInited) {
                …
                if (this.startTime === null) this.startTime = time;
                // traverse scenegraph and, for nodes having animation,
                // compute the animation matrix
                this.graph.root.computeAnimation(time - this.startTime)
        }
    }
```

Developing animation

```
class MyComponent {
    …
    init() {
        …
        this.animationMatrix = null; // no animation matrix
    }
    computeAnimation(ellapsedTime) {
        // if node does not have animation return
        // if beyond animation range, animationMatrix is the last animation state and return
        // know the active animation segment based on ellapsed time
        // calculate execution ratio [0..1] within the active segment
        // calculate TRS properties based on execution ratio
        // compute animationMatrix mat4 based on active segment start state and TRS properties
        // animationMatrix mat4 is used in the display method
    }
    …
}
```

IMPORTANT: an animation can be applied to multiple components. A componente can be referenced in multiple parts of the scenegraph. Thus, the progress of a particular animation instance for a particular component instance requires to be stored in its own instance.

Developing animation

```
class MyComponent {
    …
    display(){
        …

        // apply animation transformation if animation present
        if (this.animationMatrix !== null)
            this.scene.multMatrix(this.animationMatrix);
        …
    }
}
```