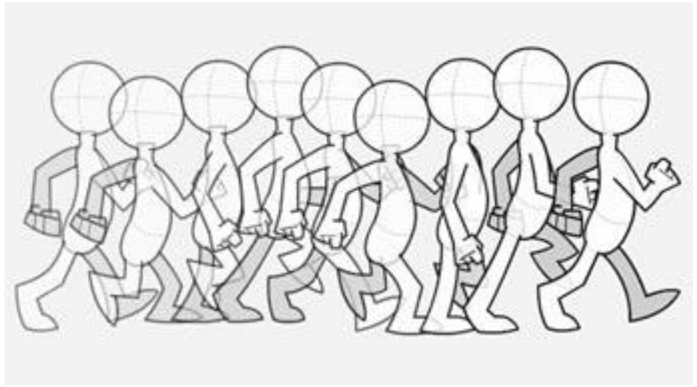
VGP334 - Animations

Instructor: Peter Chan

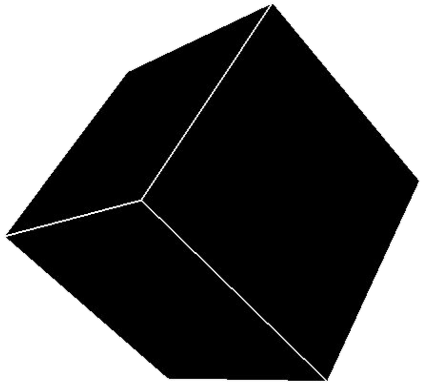
What is Computer Animation?

- Process used for generating animated images
- Can be 2D or 3D
- Successor to traditional animation techniques such as frame-by-frame animation of 2D illustrations, and stop motion using 3D models



What is Computer Animation?

- Really just a trick to the eye and the brain
- Movement is perceived by showing computer generated images with slight variations at a high frame rate (> 12 fps)



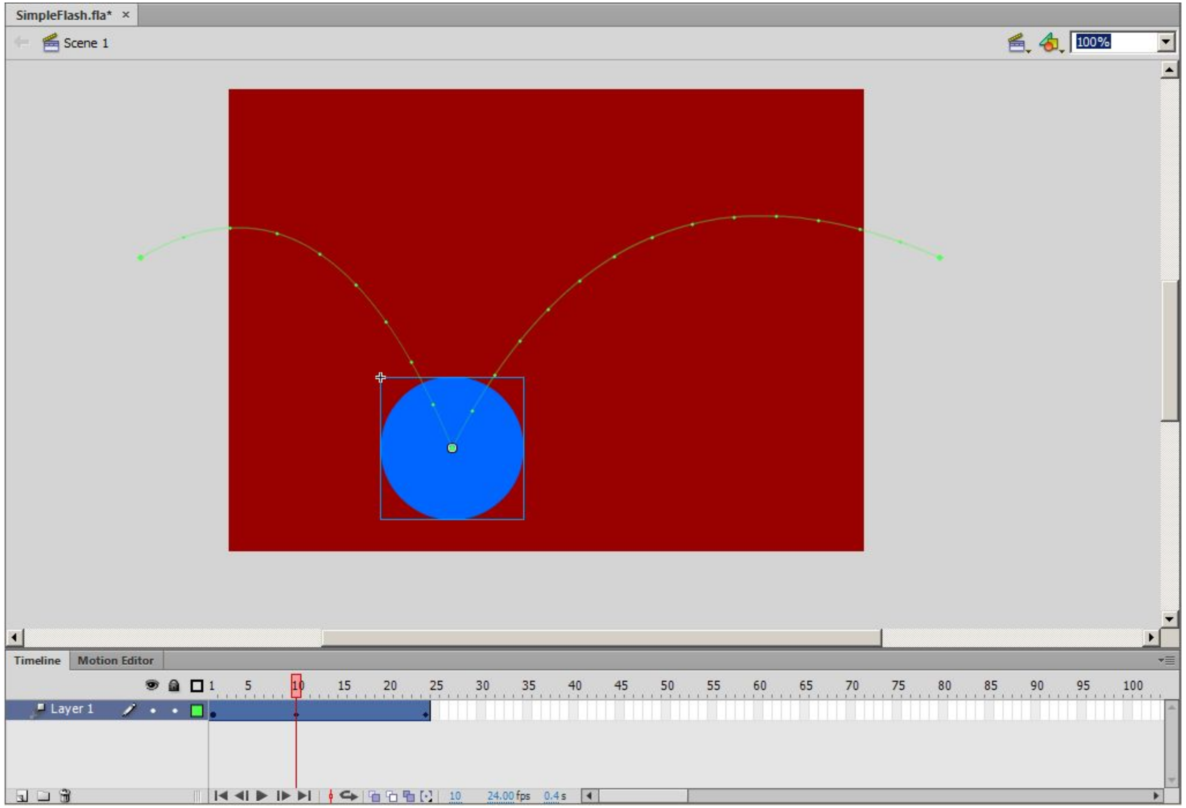
Keyframes/Tweening

- In traditional animation (think Disney), a **keyframe** is a drawing that defines the starting and ending points of any smooth transition
- Typically, it is the responsibility of a **key animator** to draw the key drawings in a scene to get across the major points of the action
- Once reviewed and approved, the keyframes are then passed on to **assistant animators** who will add details and missing frames in the scene
- Finally, the **inbetweeners** will draw in whatever frames are still missing
- This process is known as **tweening**

Keyframes/Tweening

- This concept served as the foundation for computer animation
- For example, animation tools like Adobe Animate (formerly Flash) allows you to define keyframes on a timeline
- These keyframes are used as guides for specific attributes such as position, rotation, scale, alpha transparency, etc at specific times
- The tool will then generate the in between frames procedurally using various interpolation rules during playback

Keyframes/Tweening



Mesh Morphing

- A similar idea can be applied to 3D animation as well
- For example, the popular game Quake II’s MD2 format supports animation using multiple copies of the character model
- These meshes have the same number of vertices but are positioned individually to represent the different character poses
- Tweening is achieved by interpolating the positions and normals of these meshes to dynamically create new meshes
- Since the vertex layout is the same, you can use the same index buffer to draw each frame

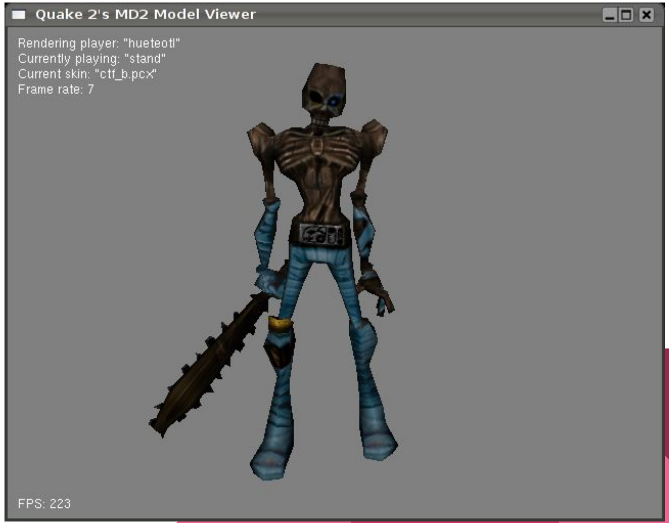
Mesh Morphing

- For a tutorial on how to read an MD2 model, you can check this link:

  http://tfc.duke.free.fr/coding/md2-specs-en.html

- Actually, this guy has a pretty good website:

  http://tfc.duke.free.fr/



```
Rendering player: "hueteotl"
Currently playing: "stand"
Current skin: "ctf_b.pcx"
Frame rate: 7

FPS: 223
```

Animation Framework

- As a first step, we will implement a fairly simple animation framework in our engine
- This framework will allow us to specify animation keyframes to control transformations including position, rotation, and scale
- Each keyframe will also have a float to represent a specific point in time
- It will be useful to control movements of static objects like a spaceship

Animation Framework

The **Keyframe** structure will look as follow:

```
struct Keyframe
{
    Math::Vector3 position;
    Math::Quaternion rotation;
    Math::Vector3 scale;
    float time;
};
```

Animation Framework

Next, you will need an **Animation** class which holds an array of keyframes, as well as helper functions to get a transformation matrix given a time value:

```
class Animation
{
public:
    Math::Matrix GetTransform(float time) const;
private:
    std::vector<Keyframe> mKeyframes
};
```

Animation Framework

The GetTransform() method needs to decide which two frames you should be interpolating from. Keep in mind that you will need to LERP position and scale, and SLERP for rotation.

You should also watch out for edge cases like out of bound time value where you may only have a single frame.

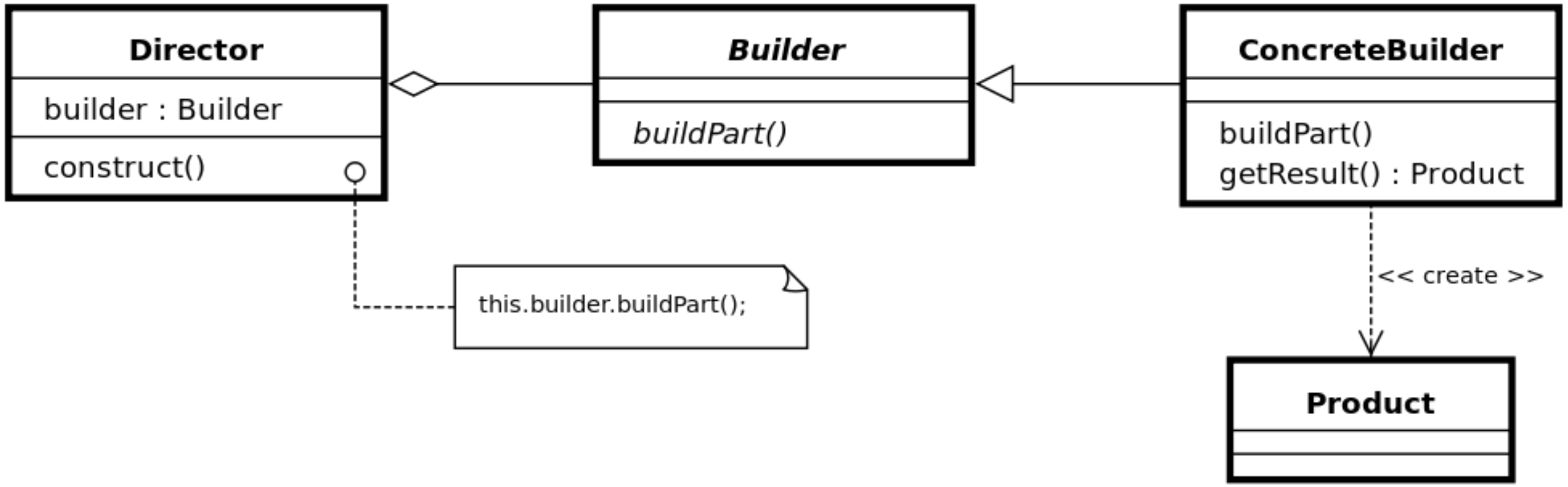
Animation Framework

Finally, you can add the **AnimationBuilder** class that allows you to create animations easily.

A good design pattern to use for this is the **Builder** pattern.

The **Builder** is a design pattern designed to provide a flexible solution to various object creation problems in object-oriented programming. The intent of the **Builder** design pattern is to separate the construction of a complex object from its representation.

Animation Framework



https://en.wikipedia.org/wiki/Builder_pattern

Animation Framework

```
class AnimationBuilder
{
public:
    AnimationBuilder& AddPositionKey(Math::Vector3, float);
    AnimationBuilder& AddRotationKey(Math::Quaternion, float);
    AnimationBuilder& AddKey(Math::Vector3, Math::Quaternion,
float);
    …
    Animation Build();
};
```

Animation Framework

What's this??

```
AnimationBuilder& AnimationBuilder::AddPositionKey(Math::Vector3, float)
{
    …
    return *this;
};
```

This is called **Method Chaining**. It allows you to invoke multiple method calls in a single statement. Just fancy syntax with no real benefit :)

https://en.wikipedia.org/wiki/Method_chaining

Animation Framework

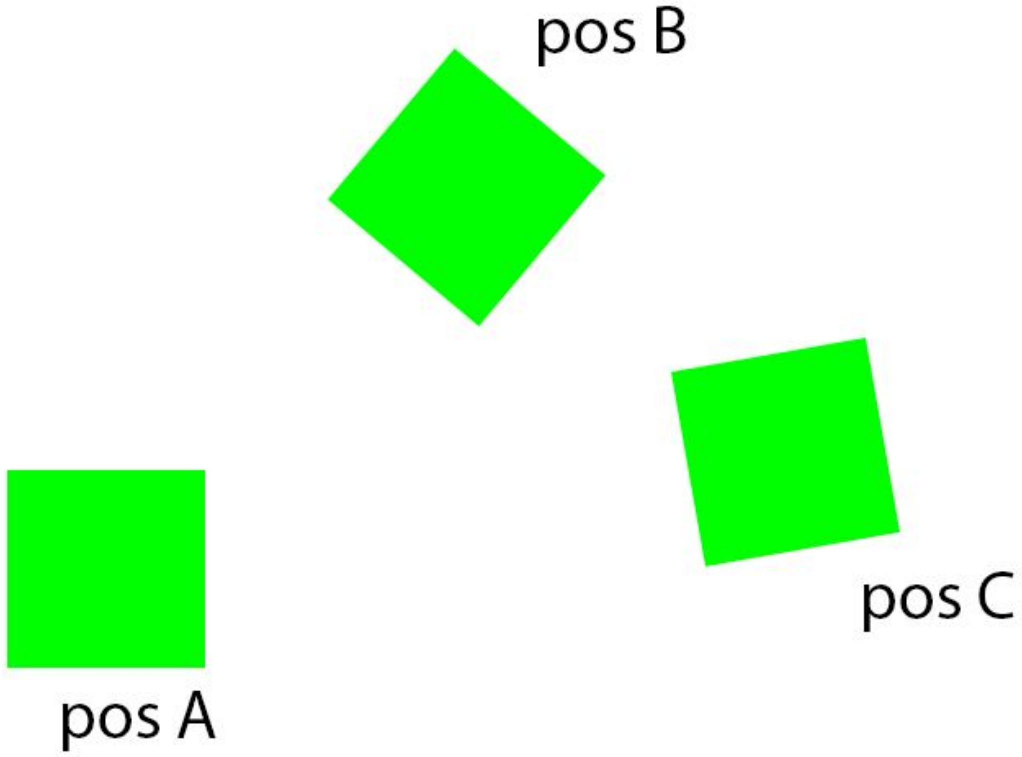
```
Animation anim =
    AnimationBuilder()
        .AddPositionKey({0,0,0},0)
        .AddRotationKey({...},1)
        .AddPositionKey({0,1,0},3)
        ...
```

Improvements

Once you have the basics working. Here are some ideas for improvements that you can try.

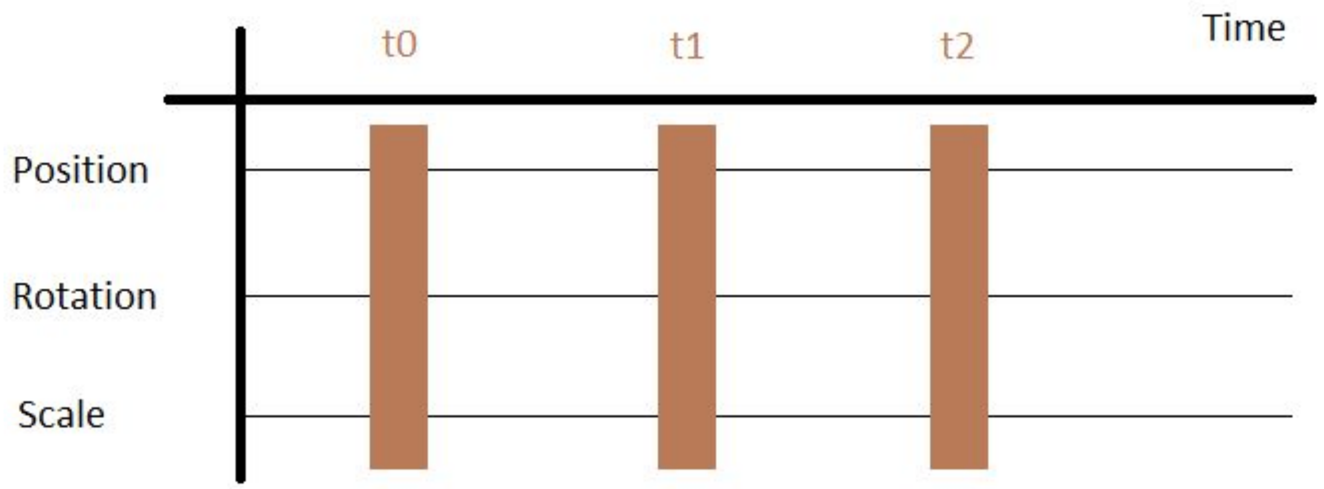
First of all, keeping all of position, rotation, and scale in a single keyframe can be wasteful. Imagine that you are animating a box, where you need to move it from position A, to position B, to position C, all while you rotation the box 80 degrees.

Improvements



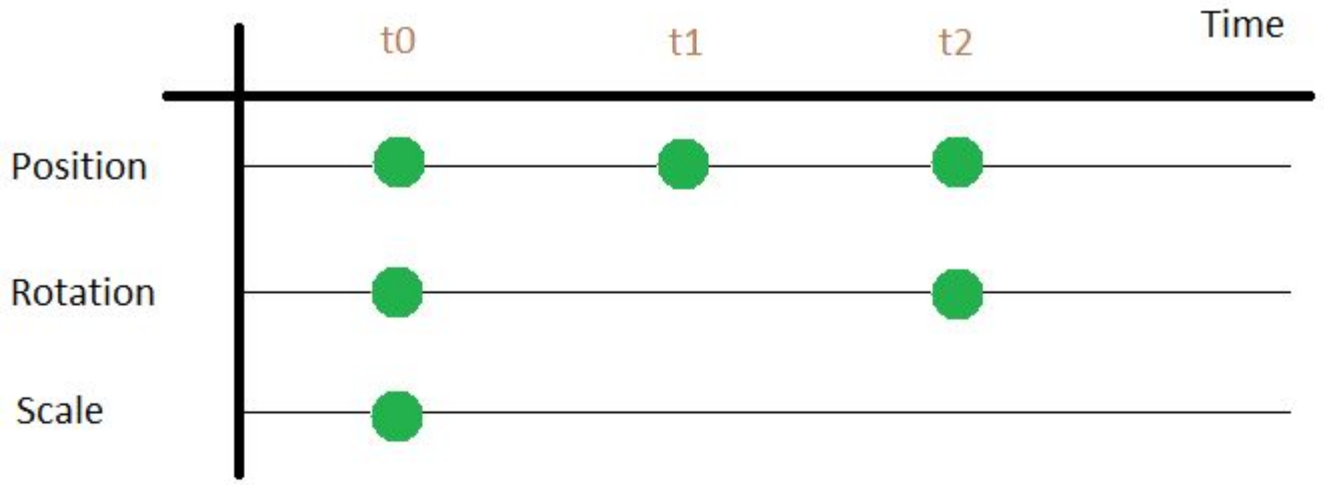
Improvements

For such an animation, you will need your keyframes as follows:



Improvements

However, technically you only really need to specify 3 positions, 2 rotation, and 1 scale keys:



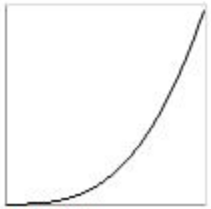
Improvements - Channels

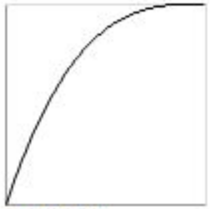
A more versatile setup will be to separate various tweening attributes into channels. The idea is that each channel can have different number of keyframes. This will save memory but will require a bit more logic to determine the correct frames to interpolate per attribute.

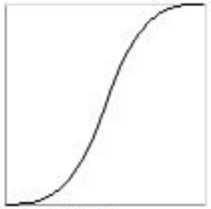
```
template <class T>
struct Keyframe
{
    T value;
    float time;
};
```

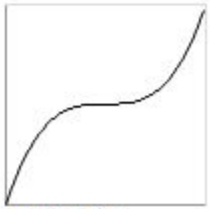
Improvements - Easing

Another feature to add to the framework is the concept of easing. Basically, instead of linearly interpolating across the keyframes, you can introduce helper functions that return warped time values:


easeIn


easeOut

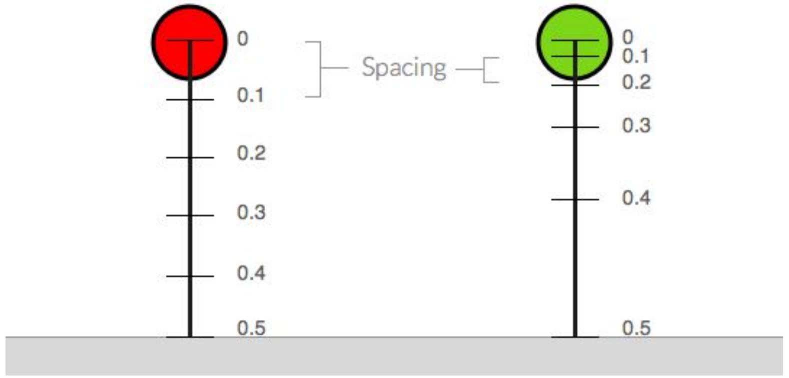

easeInOut


easeOutIn

Improvements - Easing

```
float EaseInQuad(float t)
{
    return t*t;
}

float EaseOutQuad(float t)
{
    return t*(2-t);
}
```


0
0.1
0.2
0.3
0.4
0.5
Spacing
0
0.1
0.2
0.3
0.4
0.5