Animation

Essentially a question of flipping between many still images, fast enough

---

Non Sequitur by Wiley Miller
Thursday, January 15, 2004

---

Animation as a topic

- Page flipping, double-buffering
- Sprite animation
- Movement and posing
- Collision detection and handling
- Deformations
Double buffering

Flicker-free animation

The double buffering problem

When animating a scene with many objects in real time, it is not just a question of showing images:

- Erase the entire scene
- Draw each visible object in new positions

This procedure may be visible if done on-screen!
Single buffered animation

Flicker

If the beam passes over an area while it is erased, *flicker* will occur.

**Solutions**

1) Don’t erase-and-redraw near the update beam
   
   Unreliable. Doesn’t work on all screens.

2) Double buffering.

   Needs more memory. Otherwise easy to do and reliable.
Double buffering

Choose buffer

Animation with double buffers on the video board

Fixed output image buffer

Animation with only one buffer on the video board

Double buffered animation

Tearing

If the beam passes over an area while the buffers are being switched, *tearing* will occur.
LCDs don’t have an update beam

but they have a "moment of transfer", which should still be when the image is complete.

Then "VBL" isn’t really a correct term.

Built-in VBL sync (vsync)

Modern systems have VBL sync built-in - even mandatory double buffering. You may need to turn "vsync" off to test maximum frame rate.
**Triple buffering**

A trick to reduce delays

1. Being displayed
2. Next frame, done, wait for VBL
3. Second next frame, being drawn

Don't sit and wait for VBL just because you have one image done for display!

---

**Double buffering in OpenGL**

**Double buffer**

- Pass GLUT_DOUBLE to glutInitDisplayMode
- glutSwapBuffers();

Repeated redraw in idle() or timer callback

- Update position variables
- glutPostRedisplay();
Sprite animation

An option for 2D animation.

Extremely common in games!

Hot up to 1995, now slightly less commercially interesting, but still heavily used in "multimedia" productions (Director, Flash)

Sprites in OpenGL

Use textured polygons with transparency! (Like billboards but without 3D.)

Special "blitter" calls existed in GL2, but they were not guaranteed to be fast!
Pseudo-3D effects

Scaled sprites on background with perspective:

Depth cue by **size**

Side-scrolling with parallax scroll:

Depth cue by **movement**

Depth due by **shadows**

Distance between object and shadow gives important information

Depth from shadows
**Pseudo-3D effects vs 3D**

- Depth from size = perspective projection
- Parallax scroll: Comes for free to some extent, but can be emphasized with cameras observing the viewer
- Depth from shadows: That is why shadows are important in 3D! It is needed for "full 3D" experience.

---

**Animation techniques for moving objects**

- Procedural animation
- Physics-based animation
- Pre-programmed animation paths
Animation paths

Use Catmull-Rom splines! Predictable, smooth, continuous!

Character animation

- Pre-defined poses
- Key-frame animation
- Forward kinematics
- Inverse kinematics
- Physics based animation
- Motion capture
Key-frame animation

Pre-rendered animations
Key-frames are designed at suitable intervals
Frames between keyframes are interpolated (morphed)
Very common method for real-time animation

Kinematics

Kinematics = movement without forces

Forward kinematics:
Specify poses by specifying rotation of joints. Easy to implement, but specifying poses is much trial-and-error.

Inverse kinematics:
Goal-driven posing. Specify where some part should go (i.e. a hand) and calculate necessary rotations
Motion capture

Extremely common in movies!

- Record by natural visuals only
- Tracking markers
- Active sensors on the body

Perfect for pre-generated animations.

Face animation

Hard problem - we are very sensitive to errors!

Animate by action units (muscle based) or face animation parameters (extreme detail)

FAPs part of the MPEG-4 standard.

The Candide model
Some advanced animation topics

- Bones and skinning systems
- Deformations
- Physics-based animation
- Quaternions, SLERP

Mainly subjects for later courses

Particle systems

Spectacular effects with little effort!

Many small moving objects.

- Explosions
- Water
- Fire
- Snow
- Rain
Particle system

Example: Water

No randomness - bad
Particle system

• Initial position
• Initial speed (usually with some randomness)
• Movement (usually independent, physically realistic)
• Termination rule (e.g. hits ground, fades away after some time...)

Particle system

Movement according to fundamental physics:

acceleration = gravity + forces/mass
speed = speed + acceleration
position = position + speed

“Euler integration”
Particle systems

Billboards

Lots of them!

Little use for VSD! Turn off Z-buffer!

Instancing

Special feature for drawing many simple objects
- that is, particle systems with billboards!

One call to draw a whole range of objects

`glDrawArraysInstanced`
`glDrawElementsInstanced`

`gl_InstanceID` id number in shader
Texture based particle systems

Use textures to store x, y, z, dx, dy, dz

Store as color components (r, g, b)

Needs advanced texturing features (render to texture, floating-point buffers)

Particles as billboards. Each polygon must identify its particle data.

Separate compute kernels for particle systems

CUDA, OpenCL, Compute shaders

Free choice of data formats

Less integration with the OpenGL pipeline
Basic: Start on CPU
Advanced: Go for GPU acceleration

Performance is important, but GPU based particle systems are beyond basic course goals.