Special Effects and Lighting

Like many aspects of 3D modelling (rigging, skinning, posing),
these topics are more the concerns of artists than of programmers.

Special effects (explosions, splashing water, weapon impacts, magical energy) enhance the player's experience of a game, adding beauty and interest beyond its gameplay.

Special effects (and lighting effects) can be computationally very expensive

- at some moments in gameplay it may be acceptable to devote almost all the computation time to the creation of a special effect
- mostly however the computation time has to be kept in check so that an acceptable frame rate (30, 60 frames per second or better) can be maintained.
- special effects very often involve making and moving lots of particles
- effect = visual (particle systems) plus audio and maybe tactile feedback, with an evolution over time
    - particle system = emitter plus the particles it generates

Particle System and Emitter

A particle system contains an emitter, and has a set of parameters
eg to generate a shower of sparks

- one or more textures for the sparks
- the number of sparks to be generated
- how they behave wrt gravity, air resistance
- percentage chances of sparks showing particular behaviours
    - chance they will not die when hitting the ground
    - chance they will instead bounce when hitting the ground
    - chance they will burst into more smaller sparks – generating *subparticles*

An emitter is not visible, but has shape and size. It serves to generate the particles

- how many particles
- over what spatial (cone, point, 3D mesh …) and temporal distribution
- initial positions and velocities for the particles it generates

Individual Particle

Each created particle will have parameters describing it, set when it is created

- type of particle
  - billboard, 2D sprite, 3D model, ribbon, Z-feather
- texture to be applied to it
- size
- lifetime
- speed and direction
- behaviour (e.g. die, bounce, burst)
- whether subject to physics

Particle types

- Commonest: Billboards (aka “cards”)
  - polygons in 3D always oriented toward camera: smoke often done this way
- 2D sprites
  - the particle has 3D presence but is represented by 2D directly-drawn polygon
- 3D meshes, possibly with animation
- Ribbons (aka “contrails”, “geo-trails”)
  - leave a trail behind a moving emitter (accomplished using *shaders*)
- Z-feather (aka “depth fade”; going out of fashion)
  - rendering of particle will fade off when it comes close enough to a collision

Effect types

Gameplay effects – triggered by a player action

- weapon firing, magic spell casting and consequence, impacts …

Environment effects – looping, providing ambience.

- Swirling mist, campfires, clouds …

Cinematic effects – special moments, player often losing control over game progress

- Spectacle is all-important
- Nothing much else needs to be going on simultaneously
  - hang the expense!

Destructibles – objects that can be destroyed, but not particularly spectacularly

- vehicles crashing, bottles smashing, walls crumbling …
- may need physics to get right effect

Budget for effects

If framerate of 30 or 60 fps is required, and significant time must be spent on physics and on kinematics and collision detection and AI and rendering and lighting …

- maybe 4ms, 2ms per frame (out of 33, 17) available for particle-based effects

There will be limits on what can be achieved. From Rabin's book, based on data about several game titles (Shadowrun, Halo 3, Call of Duty Modern Warfare 2)

- 3000 cpu particles
- 64 GPU particles, each can contain 512 subparticles
- 256 emitters

Lighting

Good lighting compensates for the loss of stereoscopic depth cues on the 2D screen.
It helps to define the 3D world the player perceives, to set the mood, to direct attention

- Contrast, rather than mere brightness, is what we perceive.
- (Translucency, bloom and glow are also perceptually important effects)

Film lighting typically uses three-point lighting setup

- a key light
- a fill light
- a back light ("kicker light")

This setup gives subjects a well-defined form and separateness from their environment

Types of light source

Point Light (Omni Light)

- casts light in all directions from a single point
- casts shadows (which are important for visual realism)
- attenuation (far and sometimes near) can usually be adjusted

Spotlight

- similar but casts a cone of light
- often locked on to a target object

Directional light

- casts parallel rays, as if from almost-infinite distance (eg the sun)

Area light

- simulates light being emitted from a substantial but finite area
- computationally expensive

Ambient light

- lights everything uniformly, with no particular source or direction

Baking

Lighting does not simply increase the RGB values of objects being lit.

- The proper simulation of the effects of light on the appearance of object surfaces involves calculations termed *radiosity*
- Rays of light from all sources in all appropriate directions are traced as they strike (and maybe reflect off) surfaces

Radiosity calculations are very very expensive when scene and/or light arrangements are complex

If the positions and orientations of an object and the lights are static, these radiosity calculations can be performed once off (when game is built, rather than dynamically as it is played), and the results captured as a texture to be applied to the object's surfaces. This is *baking*.

The End

but

Exam technique tips

Games Industry