

Particle Systems

CSCI 4239/5239

Advanced Computer Graphics

Spring 2017

What is it?

- Use points to create a realistic visual effect
 - Rain
 - Smoke
 - Fire
- Particle properties
 - Movement
 - Color
 - Transparency
- Can be done efficiently in a shader
 - Lots of individual points

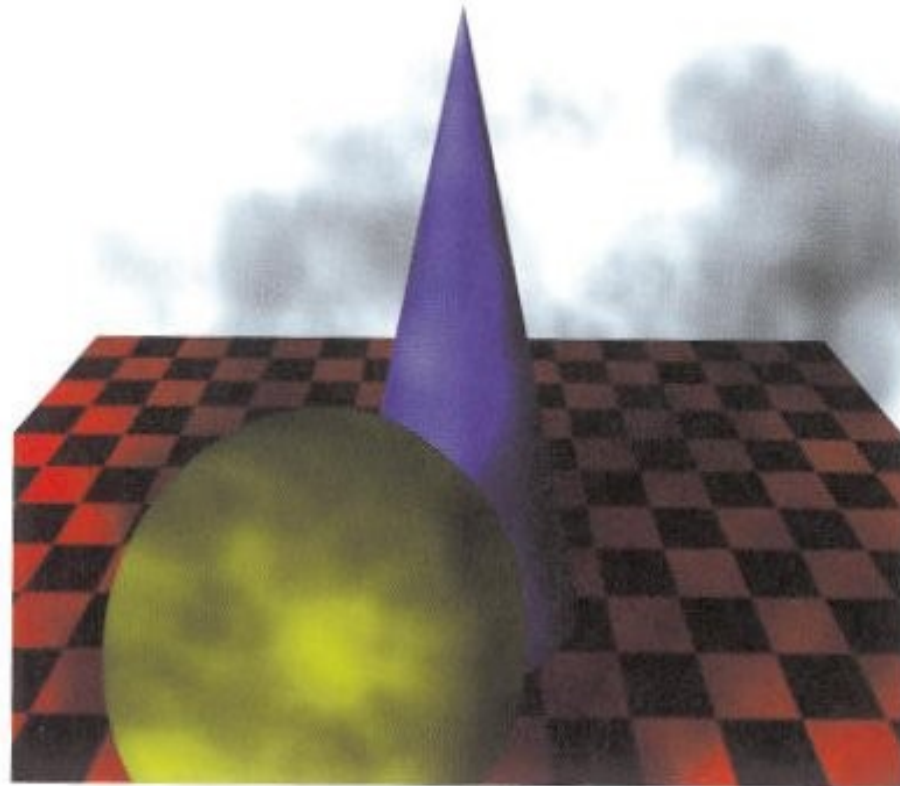
Big Particles

- Fair sized polygon
- Oriented to be face viewer
- Textured to provide details
- May be blended
- Typically uniform color
- Typically lit with constant normal

Big Particle Smoke



Single "big particle"



Smoke cloud composed of multiple big particles

From Advanced Graphics Programming Using OpenGL

Small Particles

- More suited to objects with diffuse, highly chaotic boundaries
 - Clouds
 - Fire
- One or two pixels in size, rarely textured
 - Point size may be a function of distance



From ICE Particle Shaders

Number of Particles

- Performance
- Size
- Too many may be too dense
- Too few may not be realistic
- The “best” answer is determined experimentally

Managing Particles

- Start time
- Initial properties
- Lifespan
- Behavior during lifespan
- Accumulation
- Reincarnation

Managing Particles

- Start time
- Initial properties
- Lifespan
- Behavior during lifespan
- Accumulation
- Reincarnation

Efficiently Implementing Points

- Standard attributes
 - Position, Color, Normal, Texture Coordinates
- Vertex shader may need more attributes
 - Velocity, Start time
- OpenGL Support
 - `glDrawArrays()`
 - `glBindAttribLocation()`
 - `glVertexPointer()`, `glColorPointer()`,
`glAttribPointer()`
 - `glEnableClientState()`,
`glEnableVertexAttribArray()`

Point Sprites

- Large point with a texture applied
- OpenGL Support
 - `glEnable(GL_POINT_SPRITE)`
- Vertex Shader
 - Transform particle location
- Fragment shader
 - Set every pixel on point
 - `gl_PointCoord` varies 0-1 across pixel
 - Use as texture coordinates

Particle Interactions

- Particles may or may not interact with the environment or each other
 - Sand grains bouncing off each other
 - Sand grains bouncing off an object
- Particles may be made to appear to interact by blending
 - Order and blend modes are important
- Interaction adds computational cost

CPU vs. GPU Implementation

- CPU implementation allows more flexibility
 - Access to more functions (e.g. `rand()`)
 - Significant computational costs
 - State changes are expensive (e.g. point size)
- GPU implementation
 - Huge efficiency increase
 - Parallelizable
 - A bit less flexible
 - Harder to do interactions with the environment

Applications

- Precipitation (rain, snow)
- Smoke
- Vapor trails
- Fire
- Explosions
- Cloud
- Distant lights (stars)

Application: Precipitation

- Effected by wind
- Rain drops are somewhat transparent
- Snow flakes are shiny
- Snow should accumulate
- Motion blur
- “Curtain” over scene vs. volumetric

Application: Smoke

- Smoke may rise from a source, or be everywhere in the scene
- Changes in density are a key feature
- Effected by air currents
- Turbulence
- Particles may be persistent or transitory

Application: Vapor trail

- Like smoke but fades over time
- Effected by air currents (drift)
- Generally not buoyant
- Particles typically have a finite life span

Application: Fire

- Luminescent particles
- Very dynamic
 - Buoyant
 - Turbulent
 - Short life span
- Hard to do realistically
 - Needs lots of small particles
 - Fewer large particles

Application: Explosions

- Very short lived
- Very dynamic
- Fireball
 - Centered on a point
 - Flash of light
 - Flames in Hollywood explosions
- Smoke
- Pieces of stuff blown up

Application: Clouds

- Amorphous
- Distant
- Like smoke but may be more persistent
- May be more efficient with large particles and textures using blending

Application: Light Points

- Stars
 - Twinkle due to atmospheric turbulence
 - Can be a temporal noise function
- Beacons, runway lights
 - Perspective (brightness) important depth cue
 - May be directional or periodic