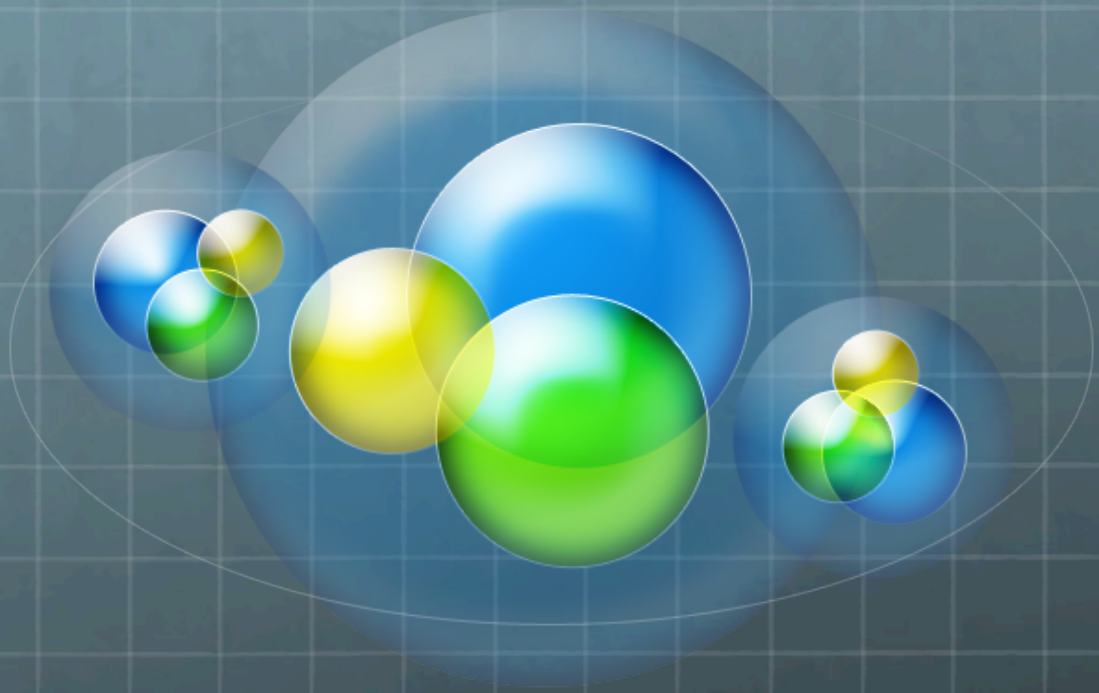


Particles

CS248 W'12

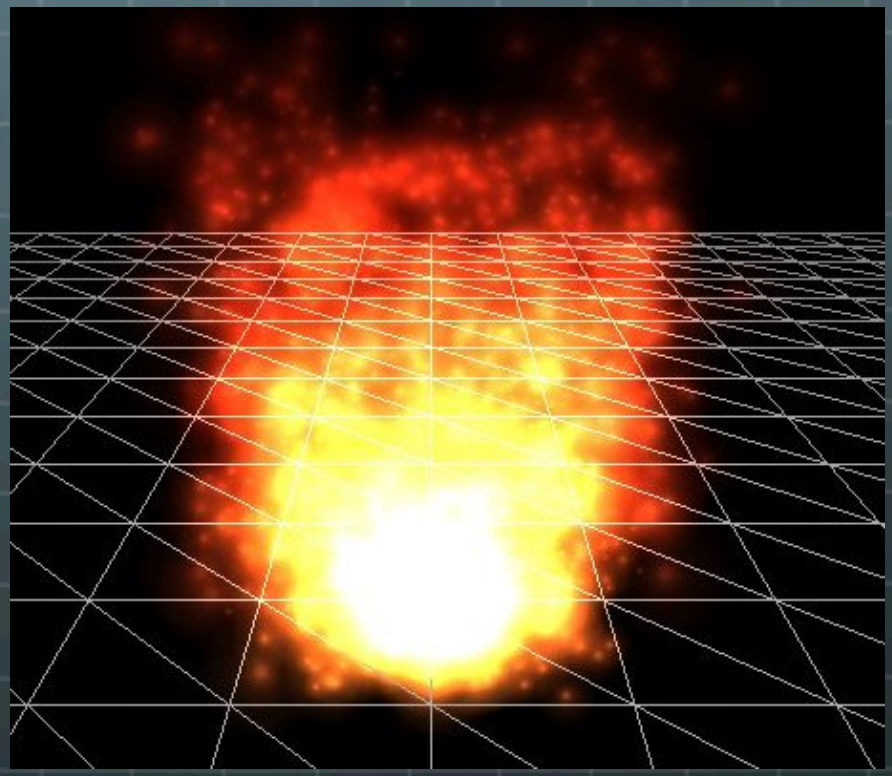
Alexander Chia

Overview

- Introduction
- Simulation (Dynamics)
  - Numerical integration
- Overview of design
  - Emitters
  - Particles + Engine
- Rendering particles

Modeling Fuzzy Phenomena

- Smoke
- Fire
- Explosions
- Rain
- Clouds



Properties of particles

- Stochastic
- Discrete, independent – work easily passed to the GPU if needed
- Some general model of behavior
  - Gravity?
  - Fuzzy movement?
  - Function of color, alpha over time

Simulation

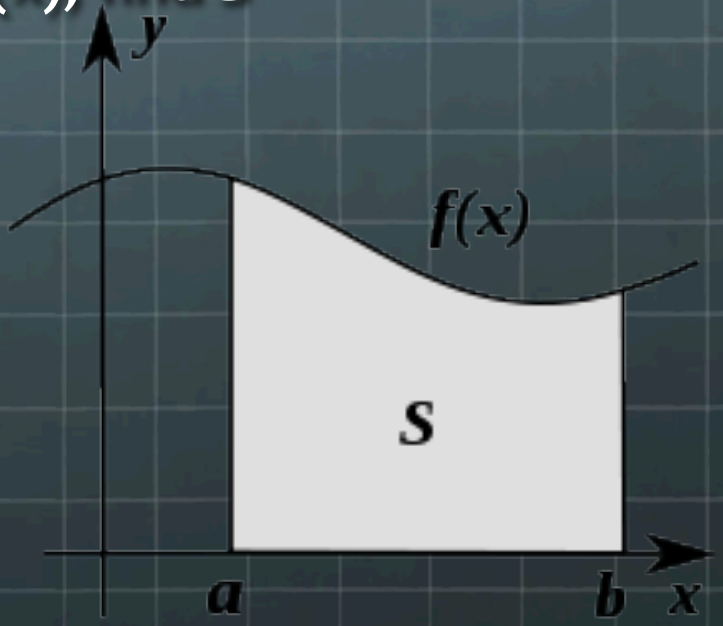
- Discrete particles acted on by forces
  - Gravity
  - Drag?
  - Collisions?
  - Attraction/Repulsion?

Dynamics

- F = ma (for constant m)
- d/dt(..) = ..’
- x’ = v
- v’ = F/m
- List of force objects
- Iterate over forces and apply to particle

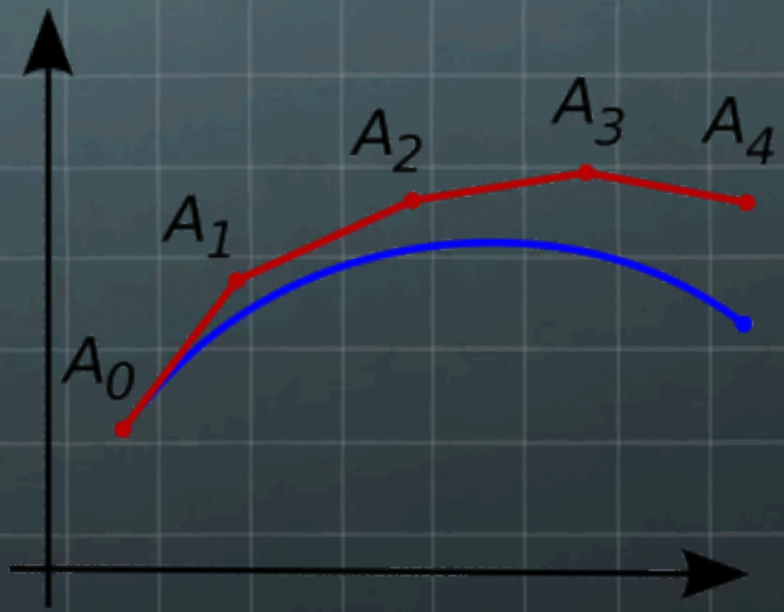
Numerical Integration

- Solve x' = f(x, t)
- Given x_0, simulate x over time
- Eg. Given f(x), find S



Euler's method

- $x(t + \Delta t) = x(t) + \Delta t\, f(x, t)$
- **Ie. at each point move along the gradient at that point**



Euler's method

- Error term dependent on size of time step

Alternative methods

- Reduce error term:
- Midpoint method
- Runge-Kutta methods

A Simple Model for particles

- $\mathbf{F} = m\mathbf{g}$
- Euler's method for update:
- $\mathbf{A}(t) = \mathbf{F}(t)/m$
- $\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \Delta t\, \mathbf{A}(t + \Delta t)$
- $\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t\, \mathbf{v}(t + \Delta t)$
- Δt = time elapsed between render() commands

Class diagram

ParticleEngine

Particle

Emitters

- Source of particles
- Spawns particles at some predefined time interval
- Sets particle params, such as initial position, velocity vector, particle lifetime etc
- Fuzzy – eg. what should the initial velocity vectors be if we want to generate particles in a cone?

Particles

- Velocity vector
- Position vector
- “Life”
- Color?
- Your_attribute_here

Particle Engine

- Controls the emitters
- Simulates the dynamics of particles across each time step
- Renders the particles

Rendering particles

- The best (and easiest) way to render particles in OpenGL is to use point sprites
- You give OpenGL some points (using `GL_POINTS` rather than `GL_TRIANGLES`)
- OpenGL automatically generates small billboards
- Enabling point sprites:
  `glEnable(GL_POINT_SPRITE)`

Point Sprite Size

- Can be set from the vertex shader
- `glEnable(GL_VERTEX_PROGRAM_POINT_SIZE)`
- Then, set the special built-in “gl_PointSize” in the vertex shader
- `gl_PointSize` = size of particle in screen units (i.e., in pixels)
- So, you need to scale by the viewport width. You also need to scale by distance from camera (length of view vector).
- `gl_PointSize = baseSize * viewportWidth / length(eyePosition.xyz)`

Transparency

- `glEnable(GL_BLEND)`
- Disable depth writes, so that particles don't block each other: `glDepthMask(GL_FALSE)`. Render particles last!
- Set the blend function: `glBlendFunc(src, dest)`. When blending, `color = src * srcColor + dest * destColor`.
- Smoke and dust (one option)
  - `src = GL_SRC_ALPHA`
  - `dest = GL_ONE_MINUS_SRC_ALPHA`
- Fire, plasma, or energetic particles
  - `src = GL_SRC_ALPHA`
  - `dest = GL_ONE`

Point Sprite Texturing

- Good news: OpenGL generates point sprite texture coordinates for you!

```
glActiveTexture(GL_TEXTURE0);
sf::Image::Bind(); or glBindTexture(id);
glTexEnvi(GL_POINT_SPRITE, GL_COORD_REPLACE,
   1);
```

- Then, just use `gl_TexCoord[0]` as the texture coordinate in your fragment shader. Simple!
- Use "splat" textures, i.e., alpha fades out from the center

Point Sprite Shader Attributes

- Each particle basically looks like this C struct:

```
struct Particle {
    Vector3 position;
    float life;
    ...maybe other parameters for cool effects
};
```

- You can pass this data to your shaders using `glVertexAttribPointer` just like you did in Assignment 3 for the tangents, normals, and positions
- Don't forget: `glGetAttribLocation`, `glEnable/DisableVertexAttribArray`, `glUniform`, `glGetUniform`, `glDrawElements`

Particle Rendering Hints

- For any `glEnable(x)` call you make before calling `glDrawElements()`, be sure you call glDisable(x) when you're done rendering the particles.
- Otherwise, weird stuff will happen to the rest of your rendering. (Or use glPushAttrib/glPopAttrib)
- Try to make your renderer general, so that there is one block of rendering code for all your particle system types.

Extensions

- Since particles are discrete, we can simulate them independently if we assume no inter-particle interactions (or if we use a simple model for that)
- GPGPU – CUDA
- Soft particles

Demo