Physics in Computer Graphics

Franklin Fang
CS 174A, UCLA, 11/13/2014

Some cool demos online

- Rigid body
- Deformable body
- Cloth
- Snow
- Smoke
- Fluid

Going real time on WebGL

- Particles
- Rigid body
- Cloth
- Water

Physics is important in Graphics

We live in a physical environment, realism in graphics essentially is physical realism. That is how things move and how things look.

Movie

- Physics Based Animation:
  - Particles, rigid body, deformable body, fluids, etc.
  - Real Time vs Offline computation.

- Physics Based Rendering:
  - Global illumination, ray-tracing, path-tracing.
  - Transparency, refraction, translucency(ios7), etc
  - Depth of field.
  - Motion blur.

Game

- Classical physics, no modern physics.
- Things are getting real time with power of GPU.

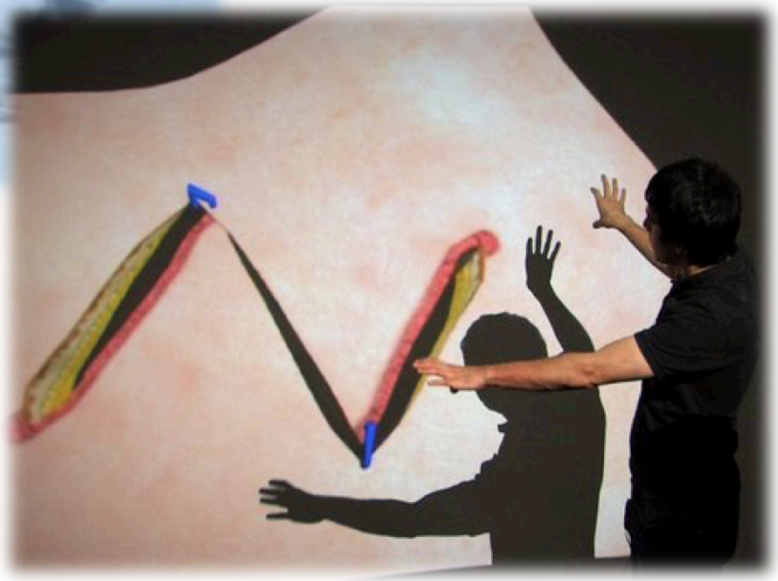
AT UCLA

Pioneering work on deformable object simulation, 1987, Demetri Terzopolous.

This work won **Academy award** for technical advancement.

http://www.cs.ucla.edu/~dt/videos/deformable-models/cloth.html

AT UCLA



Professor Joseph Teran

http://www.math.ucla.edu/~jteran/

How does it work in math

- Particles
- Rigid body
- Spring-damper System

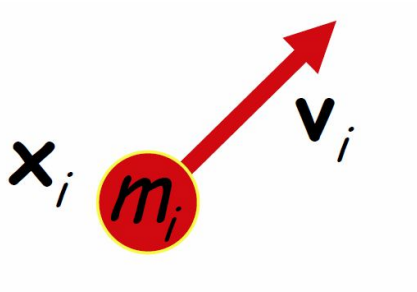
Particle Physics

The trajectory for angry birds

m_i **Mass of particle i**

x_i **Position of particle I**

v_i **Velocity of particle i**



$$f_i(t) = m_i a_i(t) = m_i \frac{dv_i}{dt} = m_i \frac{d^2 x_i}{dt^2}$$

Numerical Schemes

Forward (Explicit) Euler Method: Use current time step force and velocity

$$v_i(t+\Delta t) = v_i(t) + f_i(t) * \Delta t / m_i$$

$$x_i(t+\Delta t) = x_i(t) + v_i(t) * \Delta t$$

Backward(Implicit) Euler Method: Use future time step force and velocity

$$v_i(t+\Delta t) = v_i(t) + f_i(t+\Delta t) * \Delta t / m_i$$

$$x_i(t+\Delta t) = x_i(t) + v_i(t+\Delta t) * dt$$

No difference when force is independent of position.

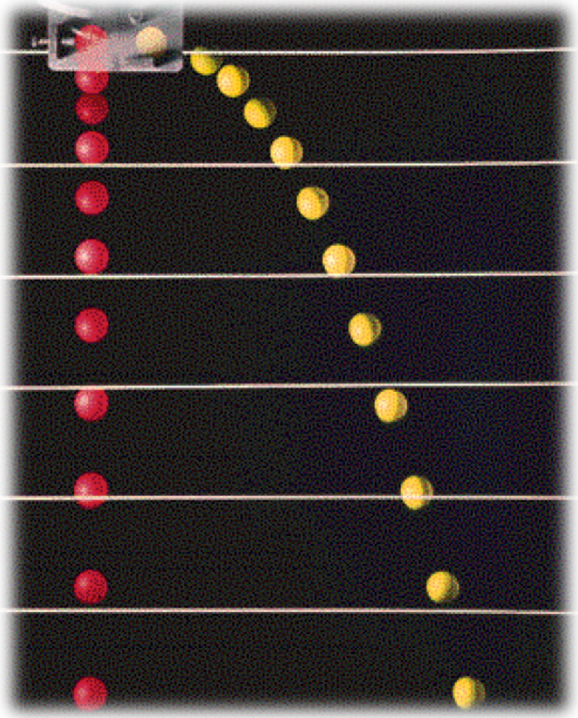
In your Code

Start

- Initiate the position, velocity and force for each particle object.

Each Frame

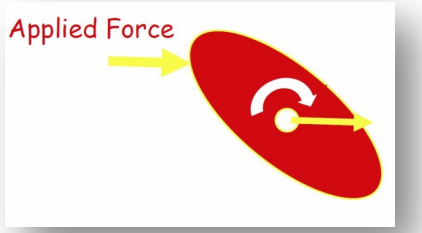
- Calculate the force, for eg. constant gravity;
- Use the force to update the velocity;
- Use the velocity to update the position;
- Draw the ball on its position.



Rigid Body

The collapse in angry bird

- Three Translational degrees of freedom: x
- Three Rotational degrees of freedom: θ



Equations of motion

$$m\mathbf{a} = \mathbf{f}$$

$$\frac{d}{dt}\mathbf{I}\boldsymbol{\omega} = \boldsymbol{\tau}$$

Torque

Angular Velocity $d\theta/dt$

$$\mathbf{I} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix}$$

where

$$I_{xx} = \int (y^2 + z^2)\,dm \qquad I_{xy} = \int xy\,dm$$

$$I_{yy} = \int (x^2 + z^2)\,dm \qquad I_{xz} = \int xz\,dm$$

$$I_{zz} = \int (x^2 + y^2)\,dm \qquad I_{yz} = \int yz\,dm$$

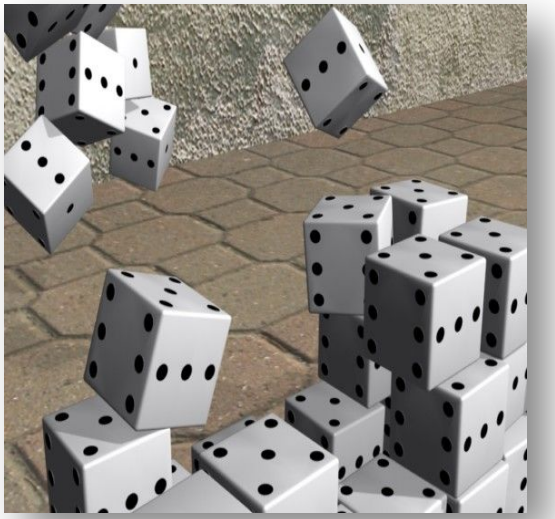
In your code

Start

- Initiate the Center of mass, Moment of Inertia, Rotation, Velocity, Angular Velocity and Force on each rigid body.

Each Frame

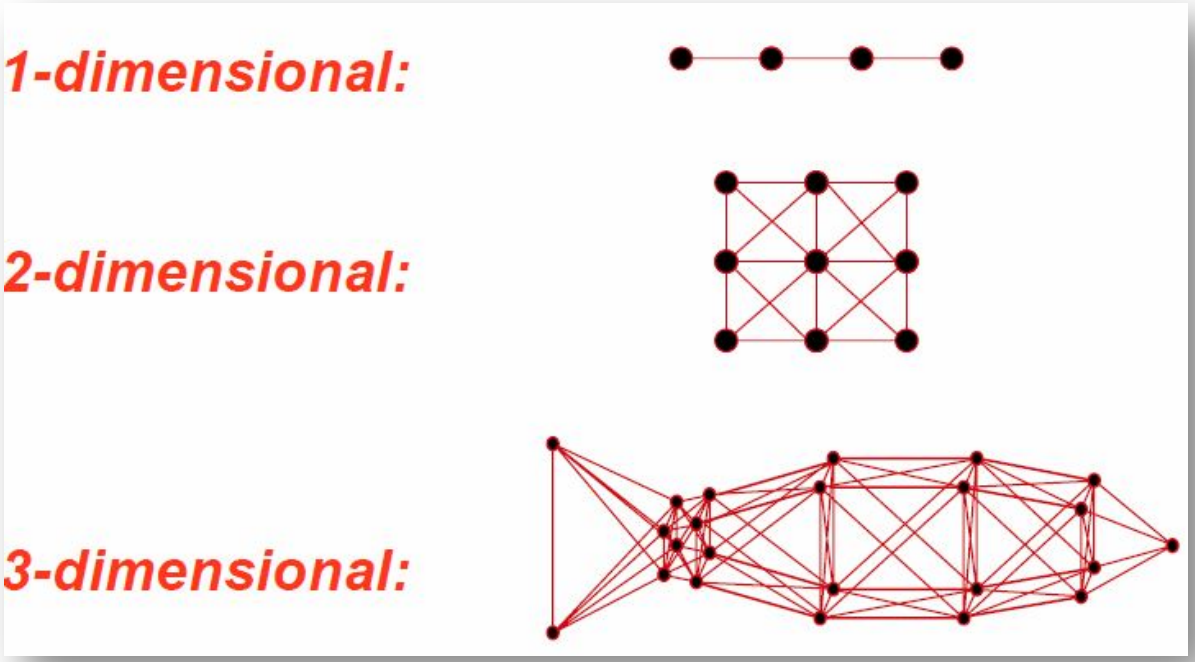
- Calculate Force and Torque (Need to solve for collision detection and collision handling)
- Update the Velocity and Angular Velocity
- Update the Position and Rotation
- Draw the object in position and rotation

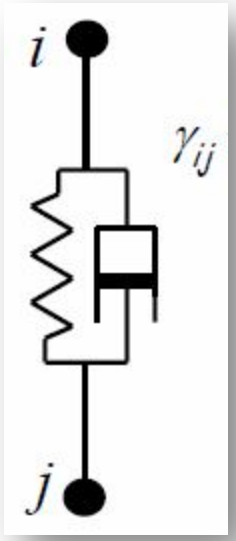


Spring-Damper System

Cloth, Hair
1D String

Spring-Damper System (The Slingshot in Angrybird)





A simple way to model deformable objects

A spring-damper connecting two particles

Spring-Damper System

Newton's law of motion

- Mass x **Acceleration** = **Net Force**
- Mathematically: for each node $i = 1, 2, \ldots, N$

$$m_i \mathbf{a}_i = \mathbf{f}_i \quad \text{or} \quad m_i \frac{d^2 \mathbf{x}_i}{dt^2} = \mathbf{f}_i$$

- This is a system of second-order ordinary differential equations in time
- The net nodal force is: $\mathbf{f}_i = \mathbf{s}_i - \gamma_i \mathbf{v}_i + \mathbf{g}_i$
  - *Gravity:* $\mathbf{g}_i$
  - *Damping force:* $-\gamma_i \mathbf{v}_i$ *(nodal drag)*
  - *Spring force:* $\mathbf{s}_i$

Spring Force

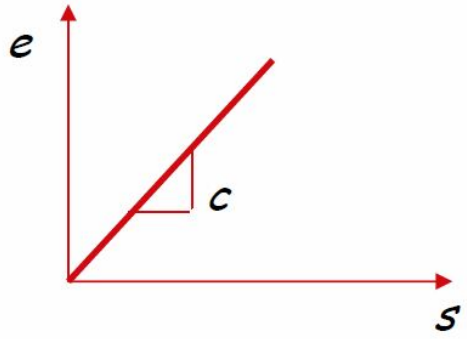
***Net spring force at node* i *is the sum of forces due to springs connecting node* i *to neighboring nodes* j**

- Denoting the neighbors of node i as N_i

$$\mathbf{s}_i(t) = \sum_{j \in N_i} \mathbf{s}_{ij}$$

Spring force

$$\mathbf{s}_{ij} = c_{ij} e_{ij} \frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|}$$



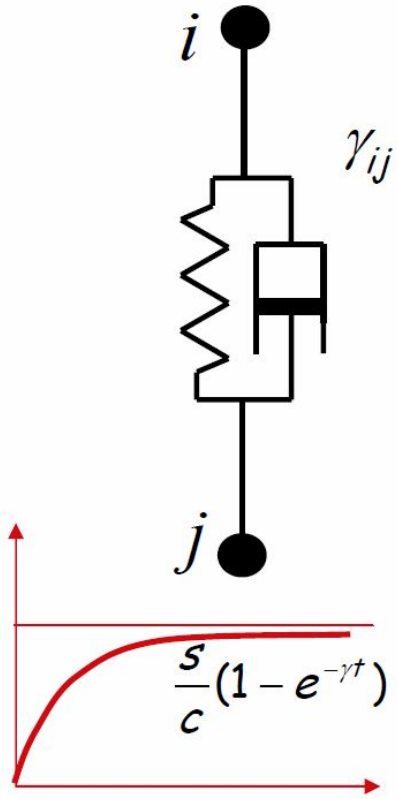
- $\mathbf{r}_{ij} = \mathbf{x}_j - \mathbf{x}_i$ is the separation of the two nodes
- $\| \mathbf{r}_{ij} \|$ is the actual length of the spring
- $e_{ij} = \| \mathbf{r}_{ij} \| - l_{ij}$ is the deformation of the spring
- Force varies linearly with deformation, but not with positions

A Damped Spring

Parallel combination of spring and damper

- Known as Voigt model
- Damping coefficient γ_{ij}

$$\mathbf{s}_{ij} = \left(c_{ij}e_{ij} - \gamma_{ij}\frac{de_{ij}}{dt}\right)\frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|}$$



Other Forces

- Friction Force (stabilize the system)
  - Friction Force is typically proportional to the speed of the node
  - Friction Factor
- External Force for manipulation
  - Need to solve the pick problem
  - Directly add to the total force on the picked node

In your Code

- First Init all the properties of nodes and spring-dampers

- Within Each Frame(Forward Euler Method)
  - Calculate and sum up all the forces
  - Update the Velocity of nodes
  - Update the Position of nodes
  - Draw everything

Numerical Schemes

Forward (Explicit) Euler Method: Use current time step force and velocity

$$v_i(t+dt) = v_i(t) + f_i(t,x) * dt / m_i$$

$$x_i(t+dt) = x_i(t) + v_i(t) * dt$$

Backward(Implicit) Euler Method: Use future time step force and velocity

$$v_i(t+dt) = v_i(t) + f_i(t+dt,x) * dt / m_i$$

$$x_i(t+dt) = x_i(t) + v_i(t+dt,x) * dt$$

Big difference when force is dependent of position of many other particles.

Advanced Deformable Object Simulation

- Demo of Jello
- Achieved by solving systems of partial differential equations
- Continuum mechanics define the equations.
- Apply advanced numerical methods to solve the equations (Finite Element Method)
- Another Eg: Professor Teran's virtual surgery.

Fluid and Smoke Simulation

- Demo
- Navier-Stokes Equations

$$\rho\left(\frac{\partial u}{\partial t} + u \cdot \nabla u\right) + \nabla P = \nu \Delta u + \rho g$$

- Usually solved by applied mathematicians

Doing physics on web?

1. How GPU shader-based simulation works
2. Cloth simulation algorithm
3. Height-field Water simulation and rendering of caustics

** Animations are created for this presentation to enhance readability. Play the presentation to get them.*

1. GPU Shader-based Simulation

Use GPU for Simulation

Why?

- Fast, 10X improvement
- Does not use CPU resource

How?

- GPGPU: general purpose computing on GPU
- Old Way: Programmable Shaders: glsl, cg, hlsl, etc
- New Way: CUDA/OpenCL

WebGL?

- Supported by Chrome/Firefox by default

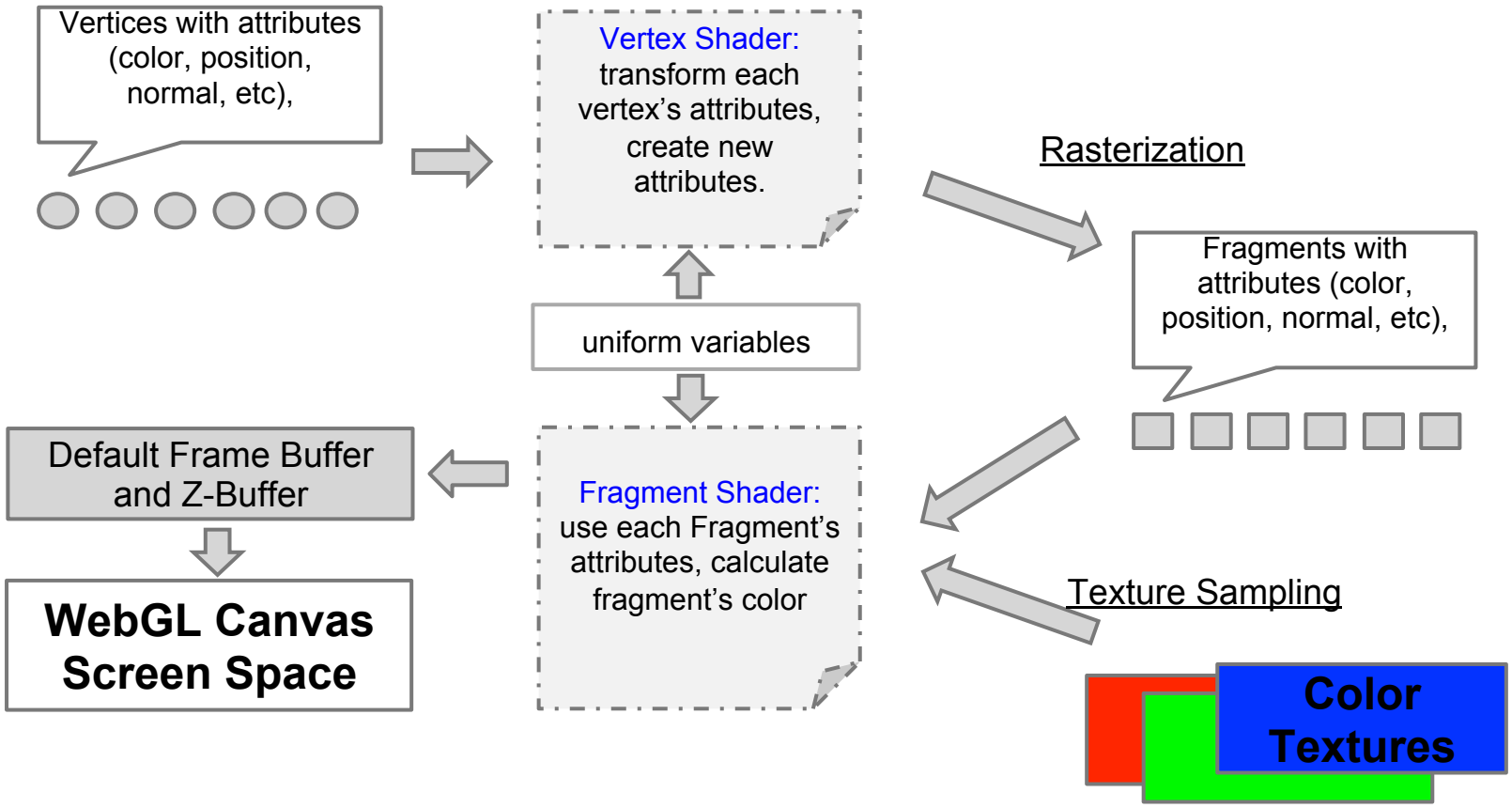
  Safari need to manually turn on

  IE is going to support soon
- Old Way: Ping-Pong data between two FBOs via shaders(glsl) to do simulation. [explained later]

WebCL?

- Still under development, it will be the better way to do it.

How shaders usually work



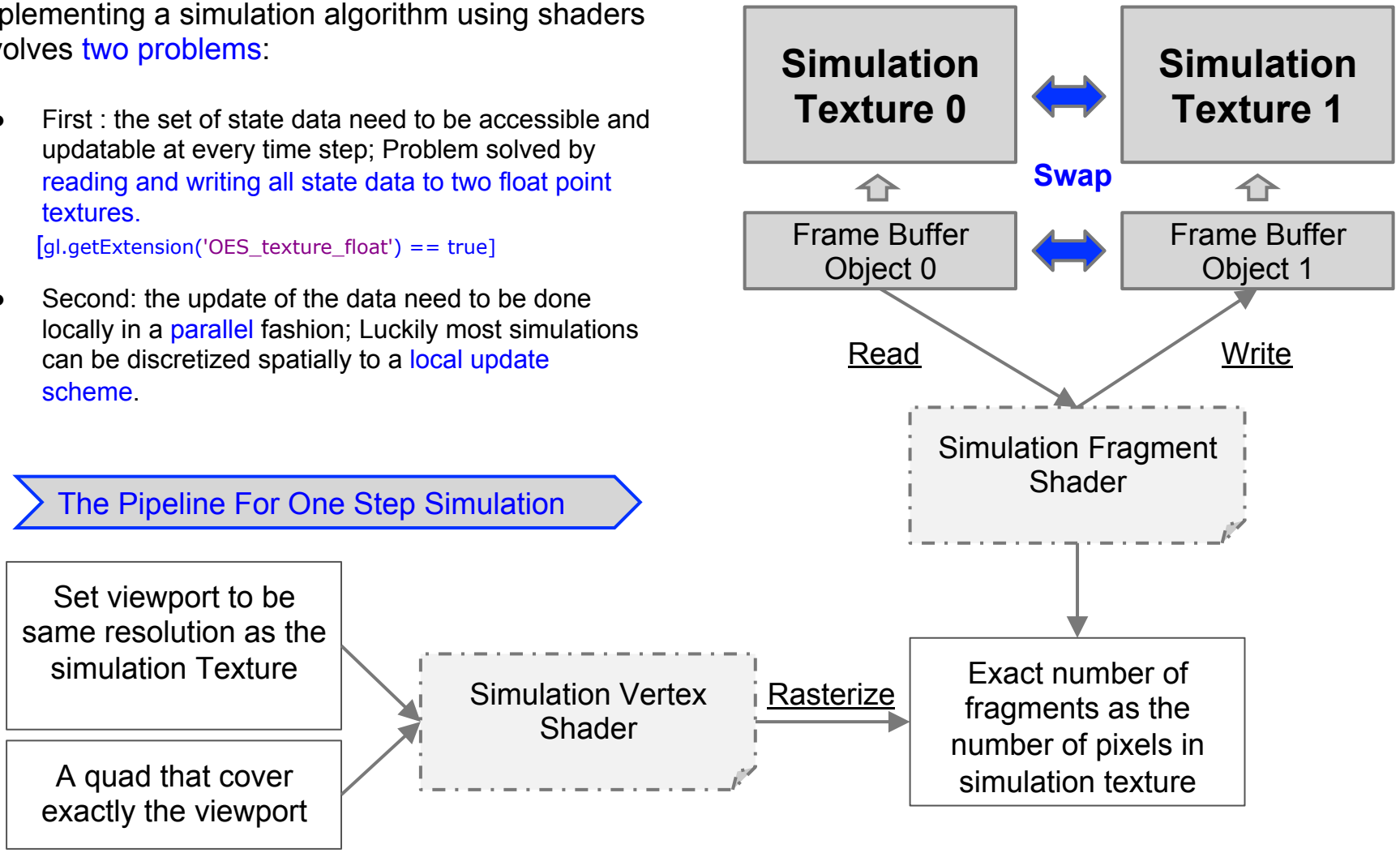
Basically:

- A pipeline that iterates at >30 frames per second;
- The vertex shader and fragment shaders are executed in parallel on many GPU processors.

How simulation shaders work

Implementing a simulation algorithm using shaders involves two problems:

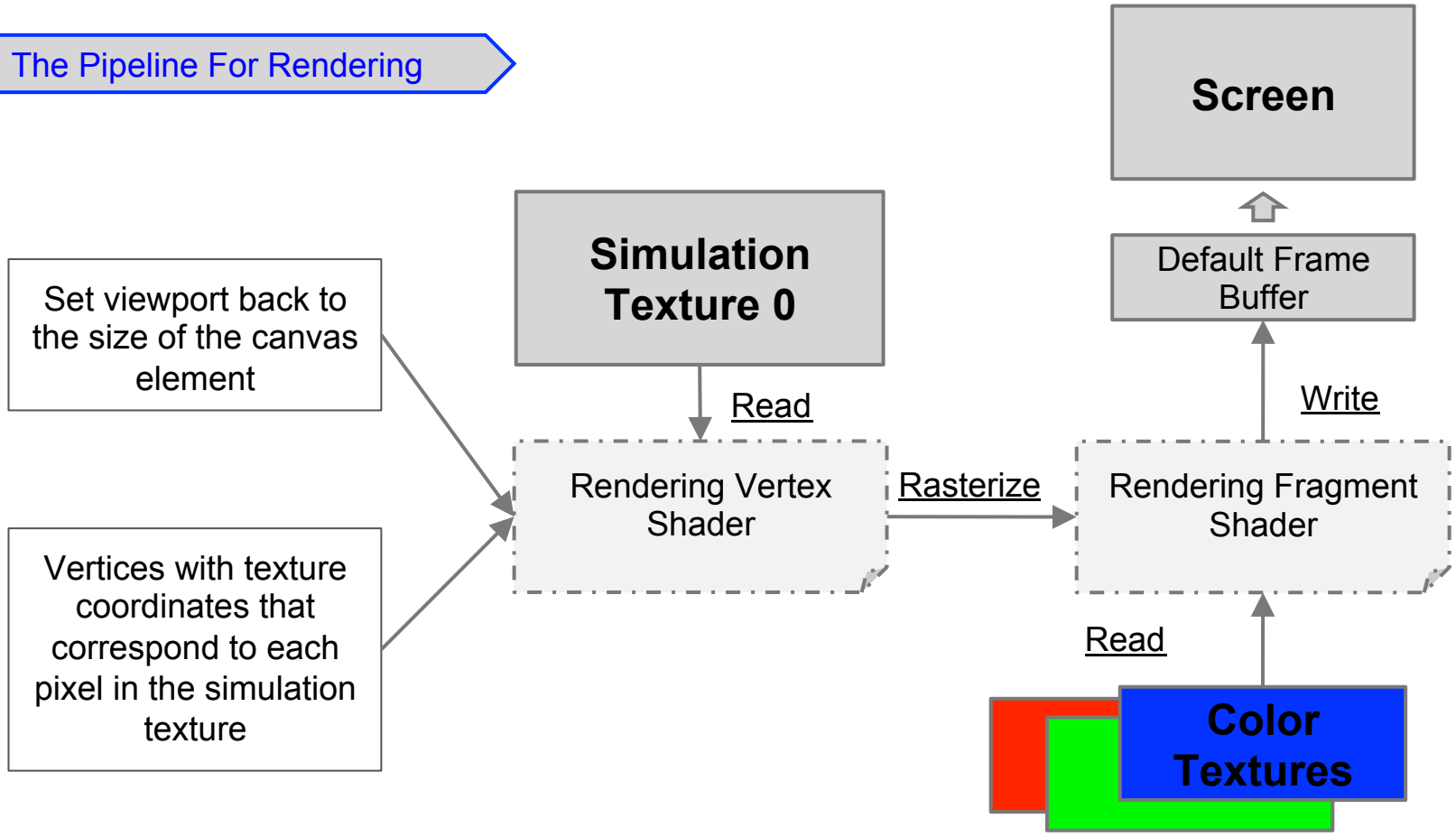
- First : the set of state data need to be accessible and updatable at every time step; Problem solved by reading and writing all state data to two float point textures.
  [gl.getExtension('OES_texture_float') == true]
- Second: the update of the data need to be done locally in a parallel fashion; Luckily most simulations can be discretized spatially to a local update scheme.



Rendering

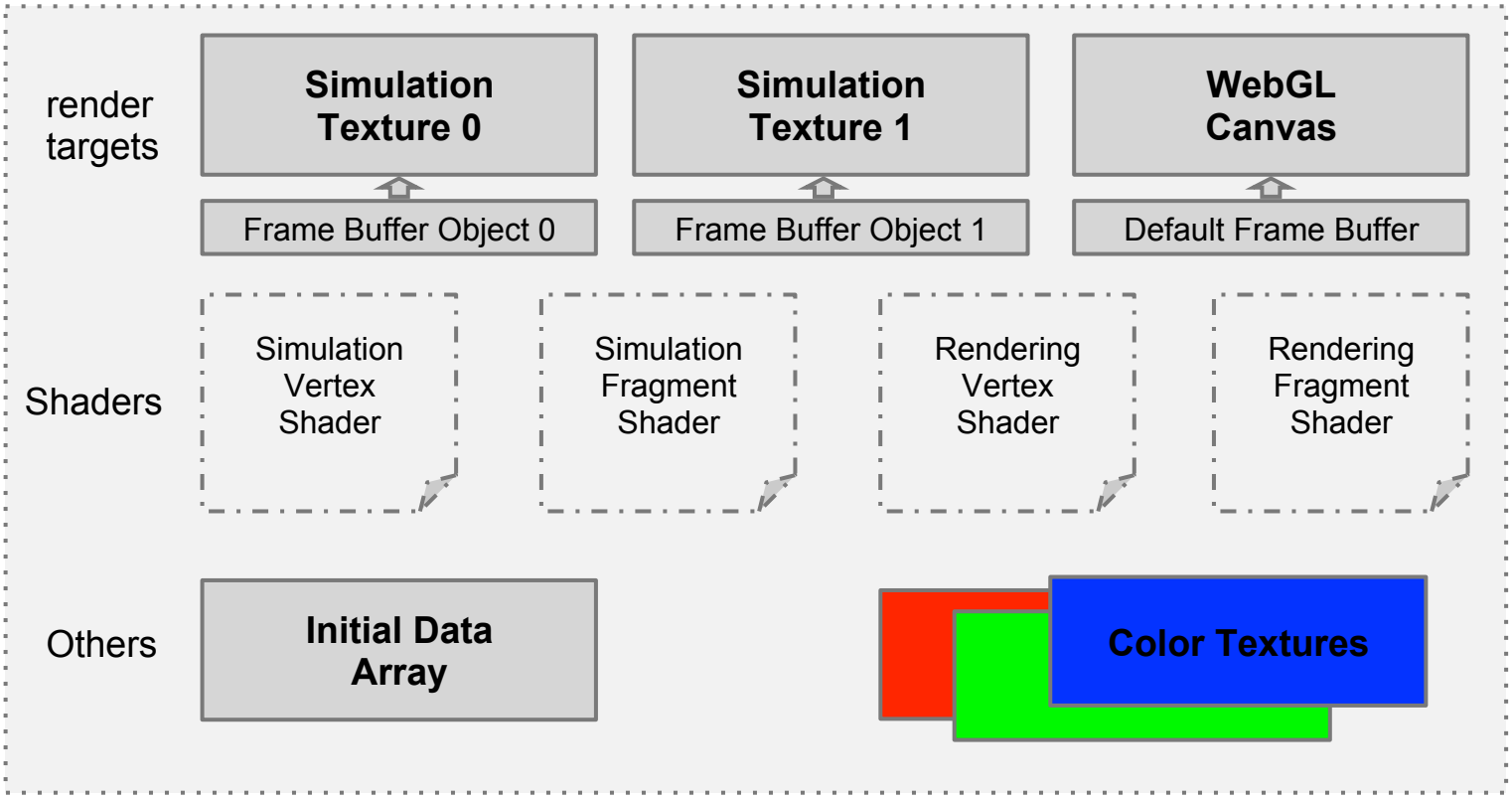
Rendering require reading data from the texture in the vertex shader.
[gl.getParameter(gl.MAX_VERTEX_TEXTURE_IMAGE_UNITS) > 0]

The Pipeline For Rendering

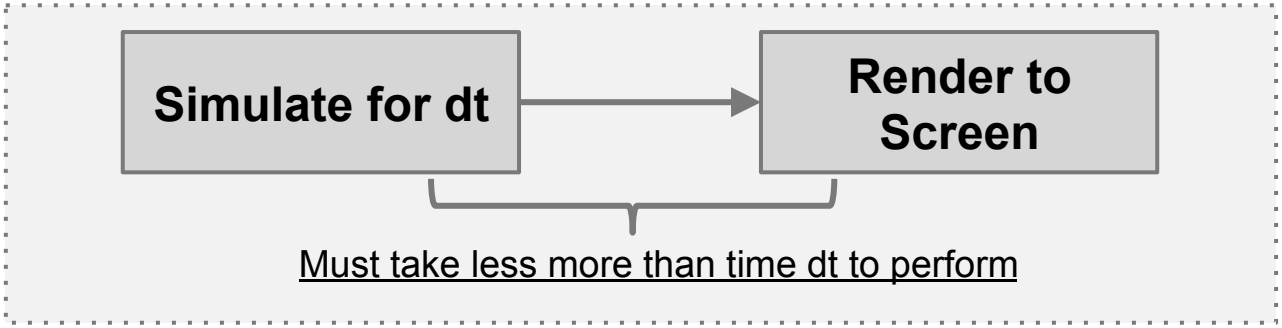


Simulation + Rendering

Initialization



Main Loop

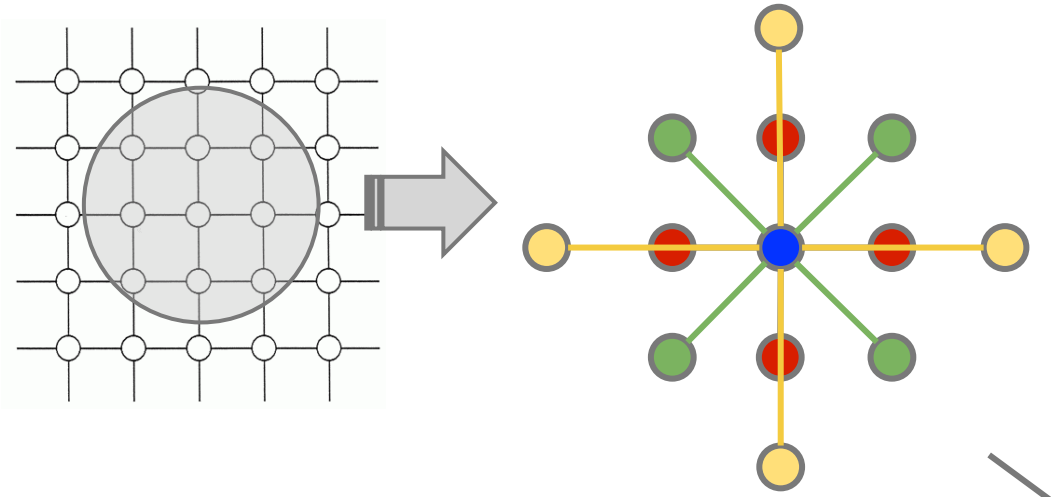


2. Cloth Simulation Algorithm

Virtual Cloth

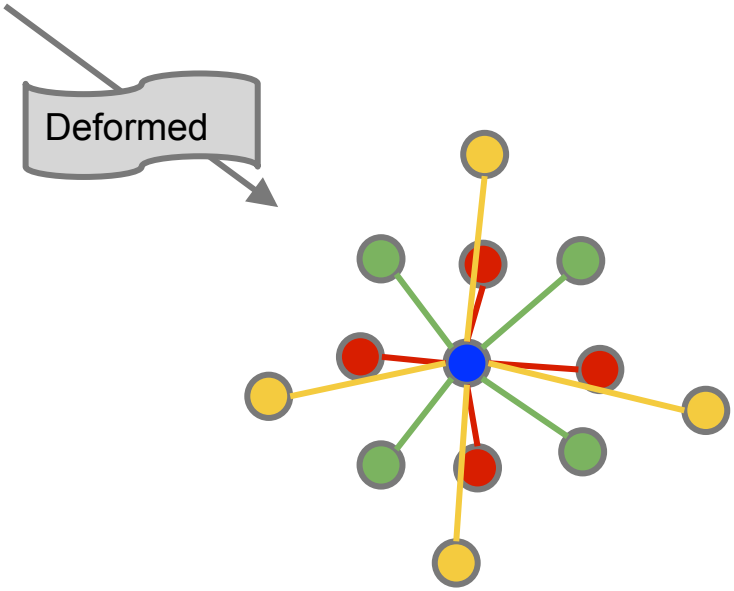
- One of the earliest physics simulation feat in graphics.
- 2D manifold embedded in 3D space, elastic material.
- Real time: Spring-Damper System + Explicit time stepping. *[What I use]*
  Offline for SFX: Continuum-Mechanics + FEM + Implicit time stepping.
- Forces: [this is just a pseudo-physical model]
  - Internal Forces: elastic force, damping force, curvature force.
  - External Forces: gravity, wind, drag, etc.

Spring Mesh Setup

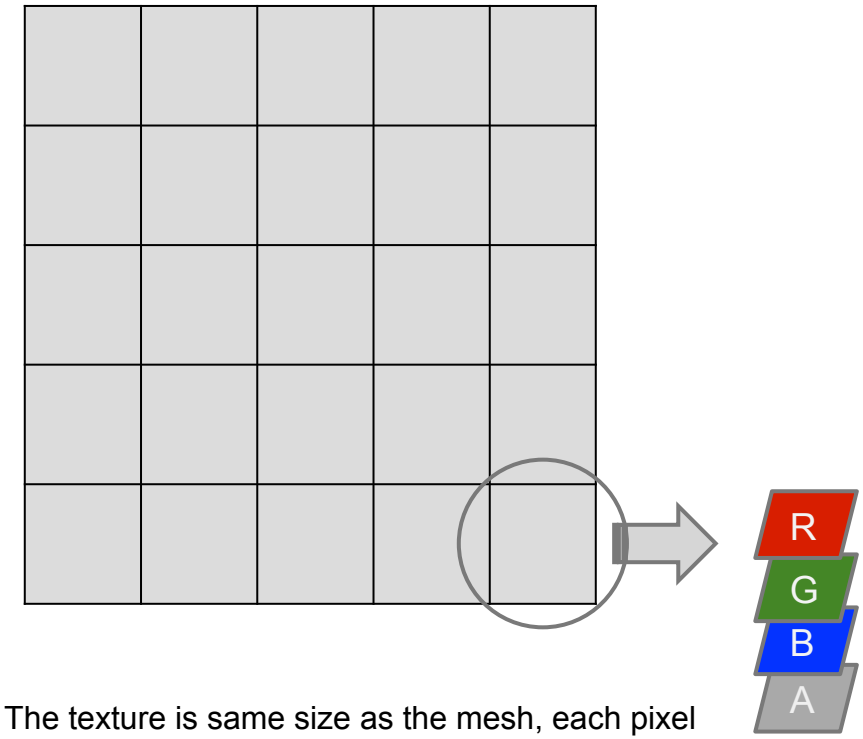


- A particular node
- Direct Neighbor
- Spring for Volume Preservation
- Spring for Curvature Energy.

- For each node, there are 12 springs connected to 12 neighbor nodes.
- The introduction of the green springs can give volume preservation effect. For eg: hanged in gravity, the sides will shrink inside.
- The introduction of the yellow springs can give curvature effect, otherwise, with only the green and red springs, the cloth can fold along the vertical and horizontal lines freely with no force pushing it back.



Data on Simulation Texture



The texture is same size as the mesh, each pixel store each node's data, such as position and velocity.

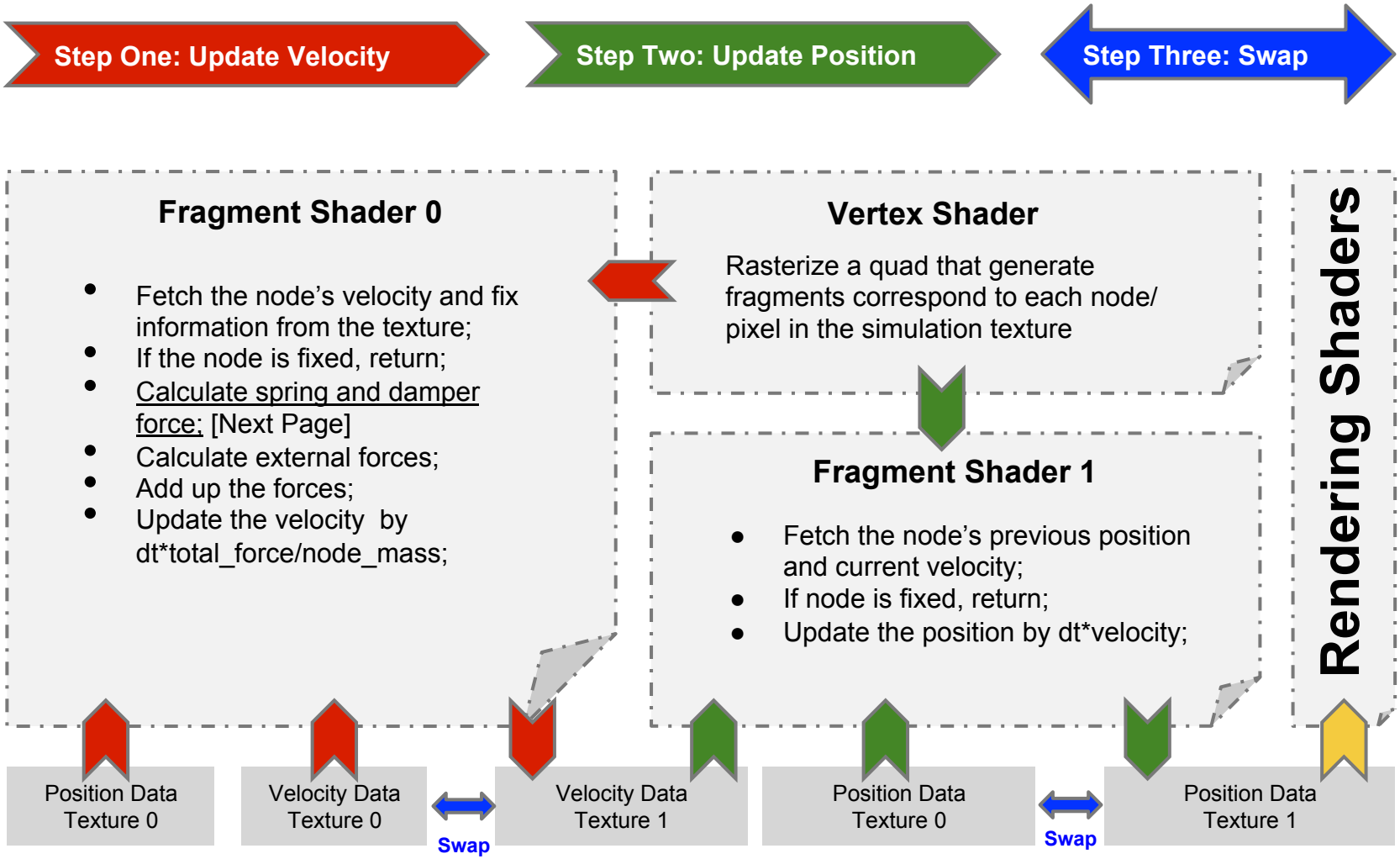
There are **two** simulation textures used to store nodal data, one for position data, the other for velocity data.

The simulation textures are float textures, each pixel can store four 32 bit floating points.

Position Data Texture
[0,1, 2]: position px, py, pz
[3]: not used.

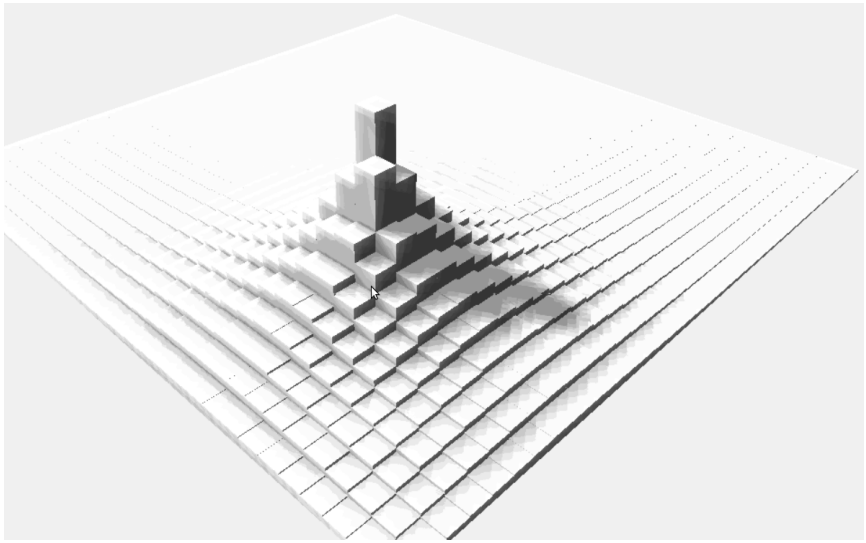
Velocity Data Texture
[0,1, 2]: velocity vx, vy, vz
[3]: set if the node is fixed or not, 0.0 means not fixed, 1.0 means fixed.

Simulation Shader Work Flow



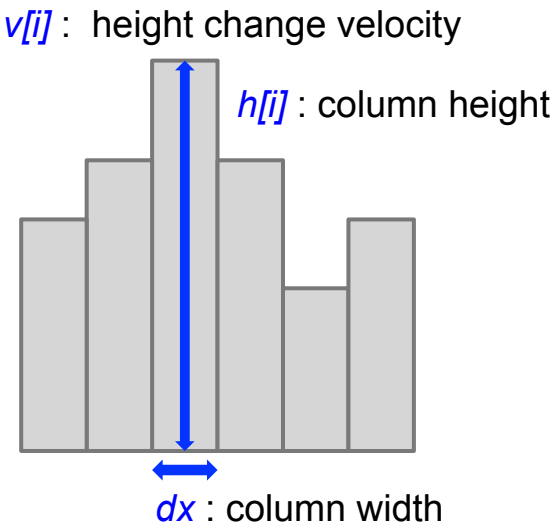
3. Height Field Water Simulation and Rendering Algorithm

Height Field Water



Represent fluid surface as a 2D function h(x,y) or a 2D array if discretized.

- Good: simplicity
- Bad: can not do breaking waves.

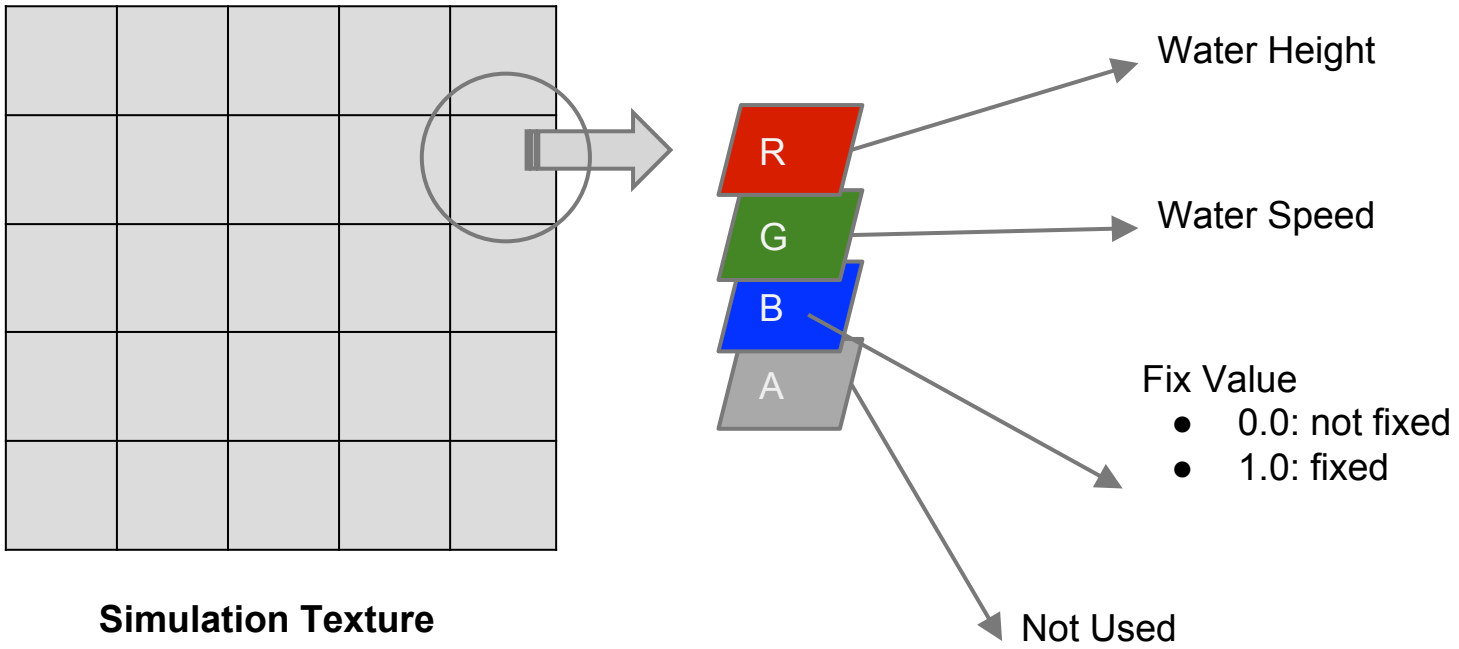


1D Example:

```
FOR each i:
    force = k*(h[i-1] + h[i+1] - 2h[i])/(dx*dx);
    v[i] = v[i] + force * dt;
    h[i] = h[i] + v[i]  * dt;
END FOR
```

k determines how fast the wave travels.

Data On Simulation Texture



Work Flow

Step One: Update Velocity and Height

Vertex Shader

Rasterize a quad that generate fragments correspond to each column/ pixel in the simulation texture

Fragment Shader

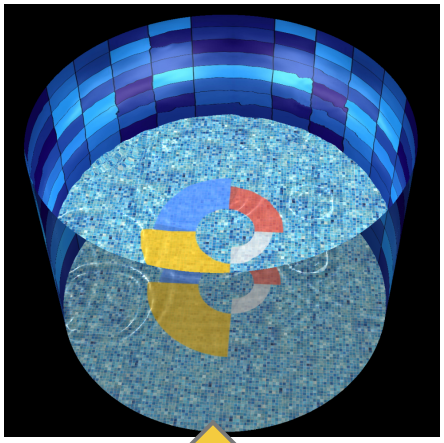
- Fetch the node's height, velocity and fix information from the texture;
- If the node is fixed, return;
- Calculate water pressure from the laplacian of water height field;
- Update the velocity;
- Update the height;

Step Two: Swap

Data Texture 0

SWAP

Data Texture 1

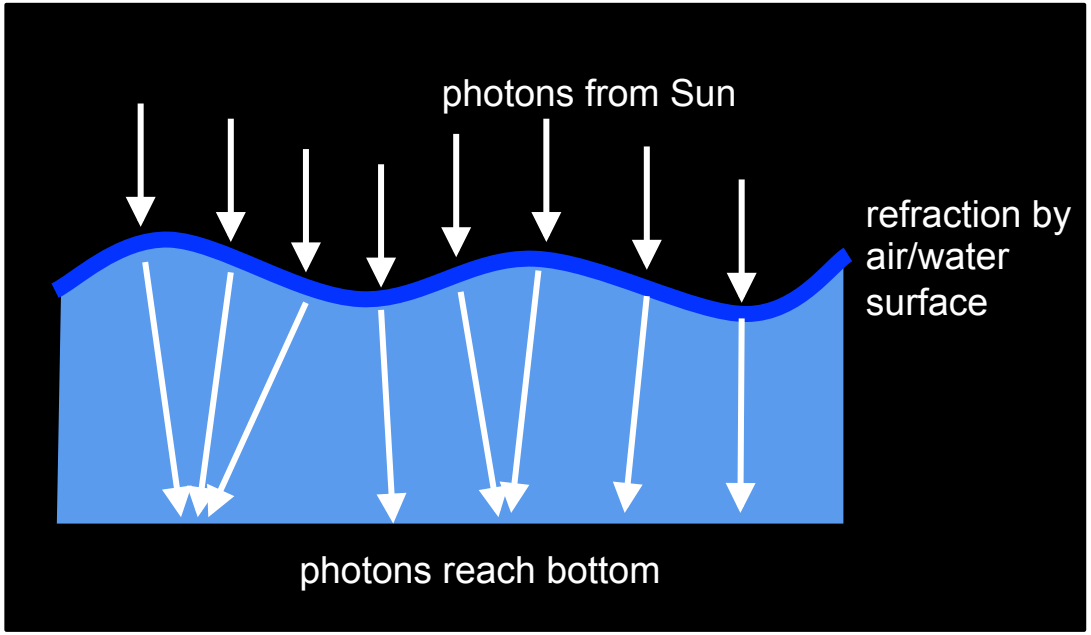


Rendering Shaders

[Next Three Slides]

Caustic Rendering

Caustics: caused by redistribution of photons via refraction through a curved surface.
Eg1: light through water surface on pool bottom;
Eg2: light through a glass on table.

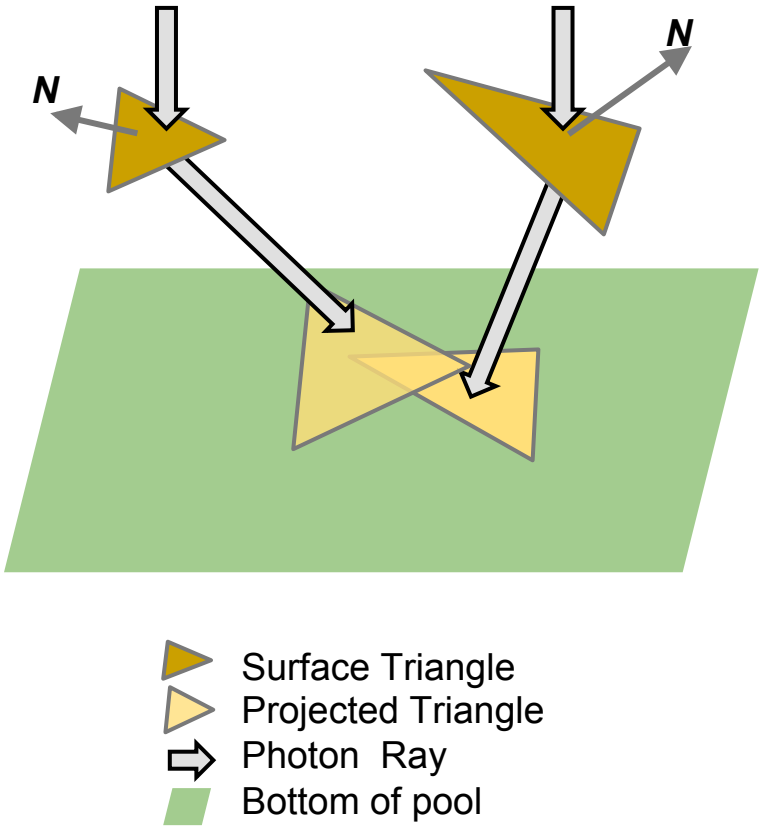


Assume the surface of caustic pattern is totally diffusive in all directions, what's needed is to generate the photon intensity map on that surface. One we have the photon map, we can simply blend that with the surface color texture to get caustic pattern.

Two-Pass Photon Map Generation

Step One:
Project surface triangles to the bottom via refraction. Also calculate the total photons from that surface triangle which is its area multiplied by cos(theta), theta is the angle between surface normal and reverse of incoming light direction.

Step Two:
Add up photon intensity from different projected triangles. To do this, turn on the blending capability and set blending to add up fragment colors. The photon intensity of each projected surface triangle is the total photons from this triangle divided by the area of the projected triangle.



Thanks for Coming!