# Stupid OpenGL Shader Tricks

Simon Green, NVIDIA

# Overview

- New OpenGL shading capabilities:
  - fragment programs
  - floating point textures
  - high level shading languages
- Make possible interesting new effects
- 2 examples:
  - Image space motion blur
  - Cloth simulation using fragment programs

# Motion Blur

- **What is motion blur?**
  - Rapidly moving objects appear to be blurred in direction of motion
- **What causes motion blur?**
  - In real cameras, film is exposed to moving scene while shutter is open
- **Why do motion blur?**
  - Avoids temporal aliasing (jerkiness)
  - Adds realism, “cinematic” look to games
  - 24fps with motion blur can look better than 60fps without

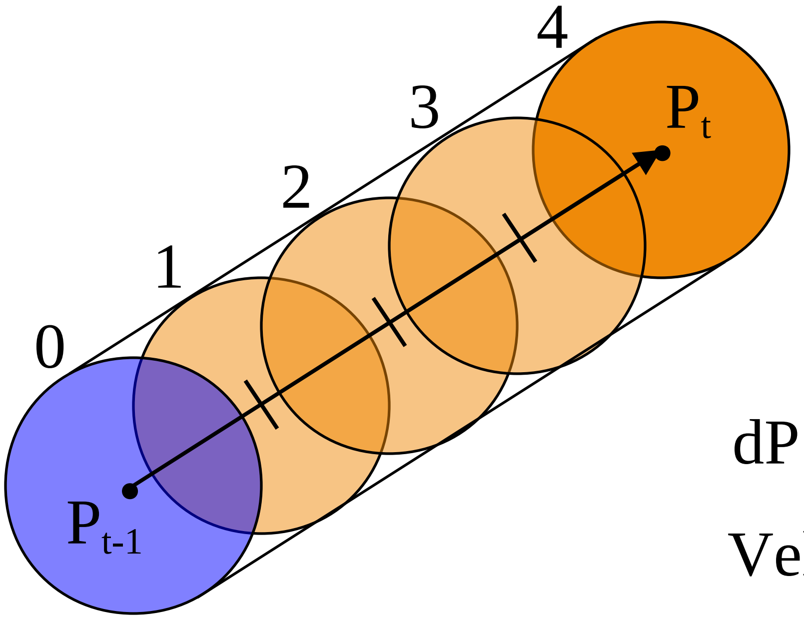
# Image Space Motion Blur

- To do motion blur correctly is hard:
  - Temporal supersampling (accumulation/T-buffer)
  - Distributed ray tracing
- Drawing trails behind objects is not the same as real motion blur
- Image space (2.5D) motion blur
  - Works as a post process (fast)
  - Blurs an image of the scene based on object velocities
  - Preserves surface detail
  - Is a commonly used shortcut in production (available in Maya, Softimage, Shake)
  - Doesn't handle occlusion well

# Algorithm

- 3 stages:
  - 1. Render scene to texture
    - At current time
  - 2. Calculate velocity at each pixel
    - Using vertex shader
    - Calculate current position – previous position
  - 3. Render motion blurred scene
    - Using fragment shader
    - Look up into scene texture
- Last two stages can be combined into a single pass

# Motion Blur




$dP = P_t - P_{t-1}$

$\text{Velocity} = dP / dt$

$P_{sample} = P + dP * u$

$N_{samples} = 5$

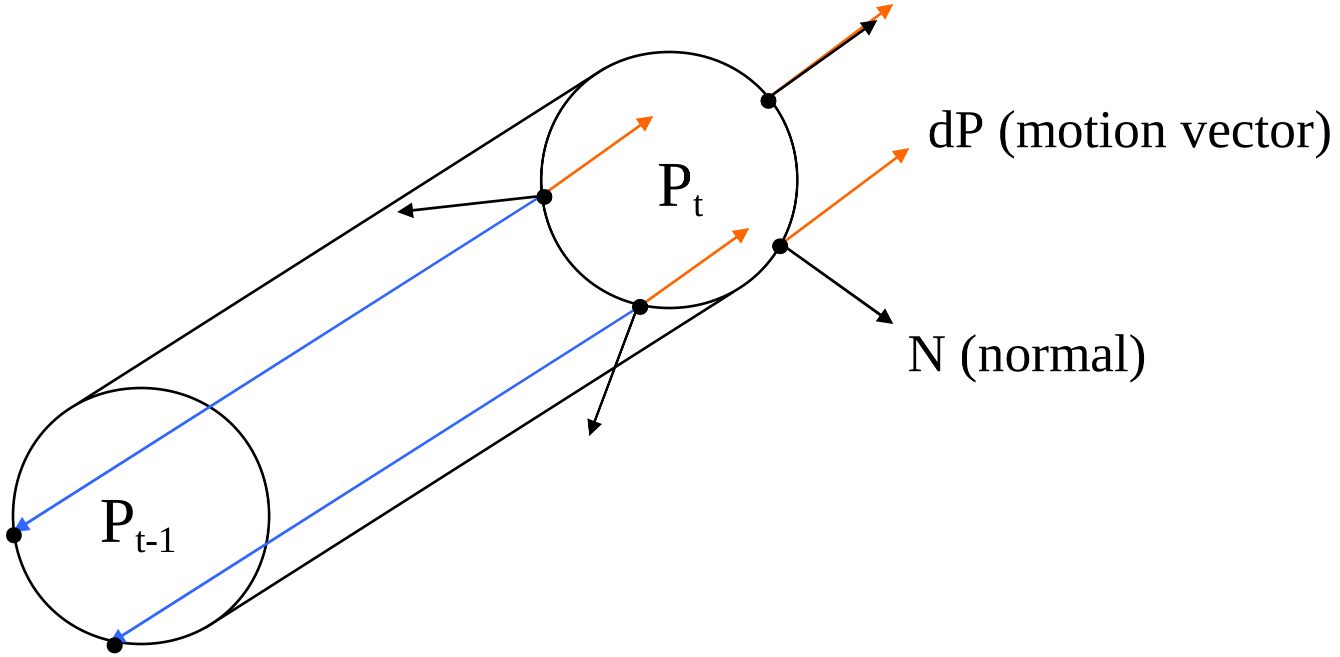
# Calculating Velocities

- We need to know the window space velocity of each pixel on the screen
- Reverse of "optical flow" problem in image processing
- Easy to calculate in vertex shader
  - transform each vertex to window coordinates by current and previous transform
  - for skinning / deformation, need to do all calculations twice
  - Velocity = (current_pos – previous_pos) / dt
- Velocity is interpolated across triangles
- Can render to float/color buffer, or use directly

# Calculating Velocities (2)

- Problem: velocity outside silhouette of object is zero (= no blur)
- Solution: use Matthias Wloka's trick to stretch object geometry between previous and current position
- Compare normal direction with motion vector using dot product
- If normal is pointing in direction of motion, transform vertex by current transform, else transform it by the previous transform
- Not perfect, but it works

# Geometry Stretching



# Vertex Shader Code

```
struct a2v {
  float4 coord;
  float4 prevCoord;
  float3 normal;
  float2 texture;
};
struct v2f {
  float4 hpos     : HPOS;
  float3 velocity : TEX0;
};

v2f main(a2v in,
         uniform float4x4 modelView,
         uniform float4x4 prevModelView,
         uniform float4x4 modelViewProj,
         uniform float4x4 prevModelViewProj,
         uniform float3   halfWinSize,
         )
{
  v2f out;

  // transform previous and current pos to eye space
  float4 P = mul(modelView, in.coord);
  float4 Pprev = mul(prevModelView, in.prevCoord);

  // transform normal to eye space
  float3 N = vecMul(modelView, in.normal);
```

```
  // calculate eye space motion vector
  float3 motionVector = P.xyz - Pprev.xyz;

  // calculate clip space motion vector
  P = mul(modelViewProj, in.coord);
  Pprev = mul(prevModelViewProj, in.prevCoord);

  // choose previous or current position based
  // on dot product between motion vector and normal
  float flag = dot(motionVector, N) > 0;
  float4 Pstretch = flag ? P : Pprev;
  out.hpos = Pstretch;

  // do divide by W -> NDC coordinates
  P.xyz = P.xyz / P.w;
  Pprev.xyz = Pprev.xyz / Pprev.w;
  Pstretch.xyz = Pstretch.xyz / Pstretch.w;

  // calculate window space velocity
  float3 dP = halfWinSize.xyz * (P.xyz - Pprev.xyz);

  out.velocity = dP;
  return v2f;
}
```

# Motion Blur Shader

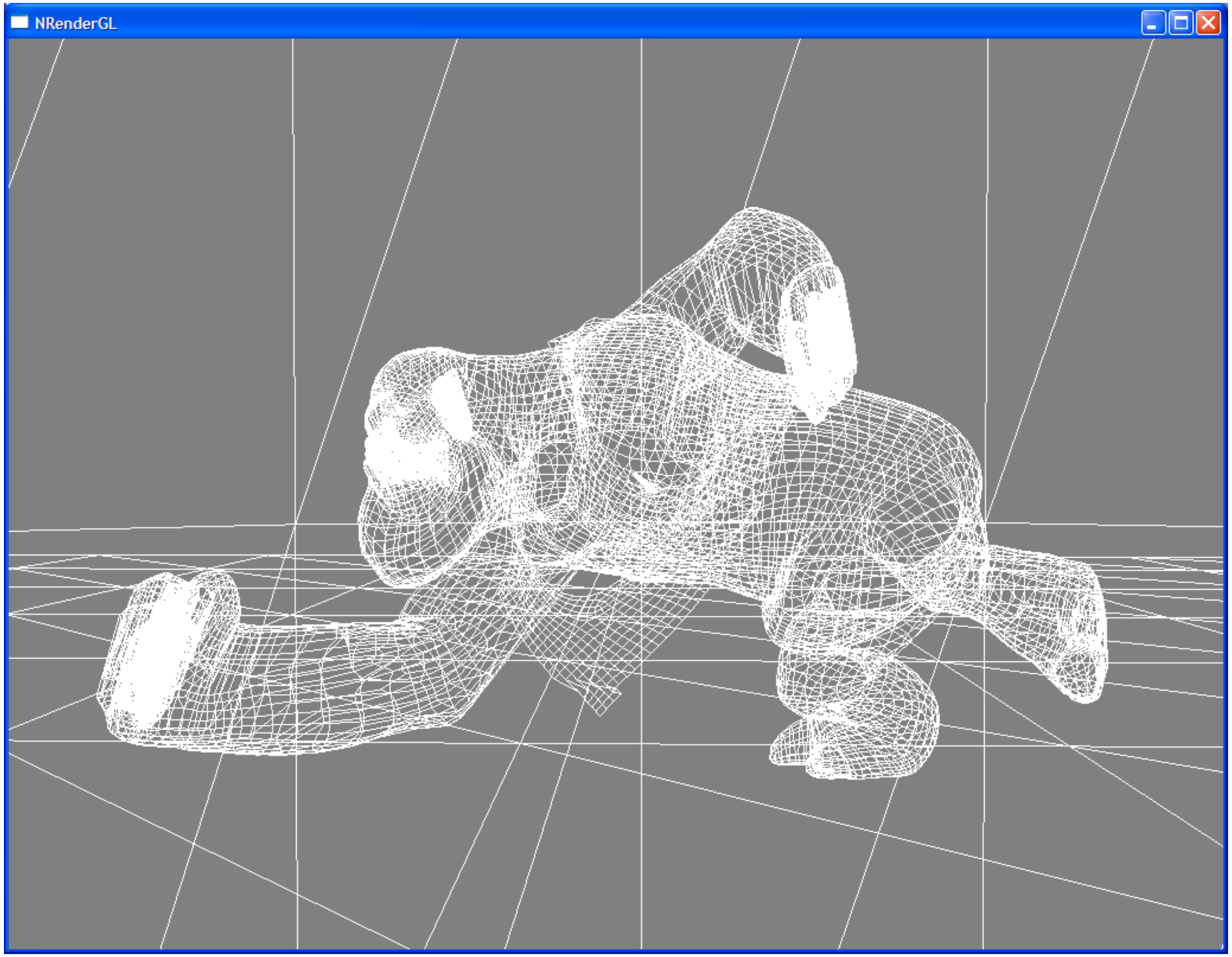
- Looks up into scene texture multiple times based on motion vector
- Result is weighted sum of samples
  - Can use equal weights (box filter), Gaussian or emphasise end of motion (ramp)
- Number of samples needed depends on amount of motion
  - 8 samples is good, 16 is better
  - Ironically, more samples will reduce frame rate, and therefore increase motion magnitude
- Effectively we are using velocity information to recreate approximate in-between frames

# Motion Blur Shader Code

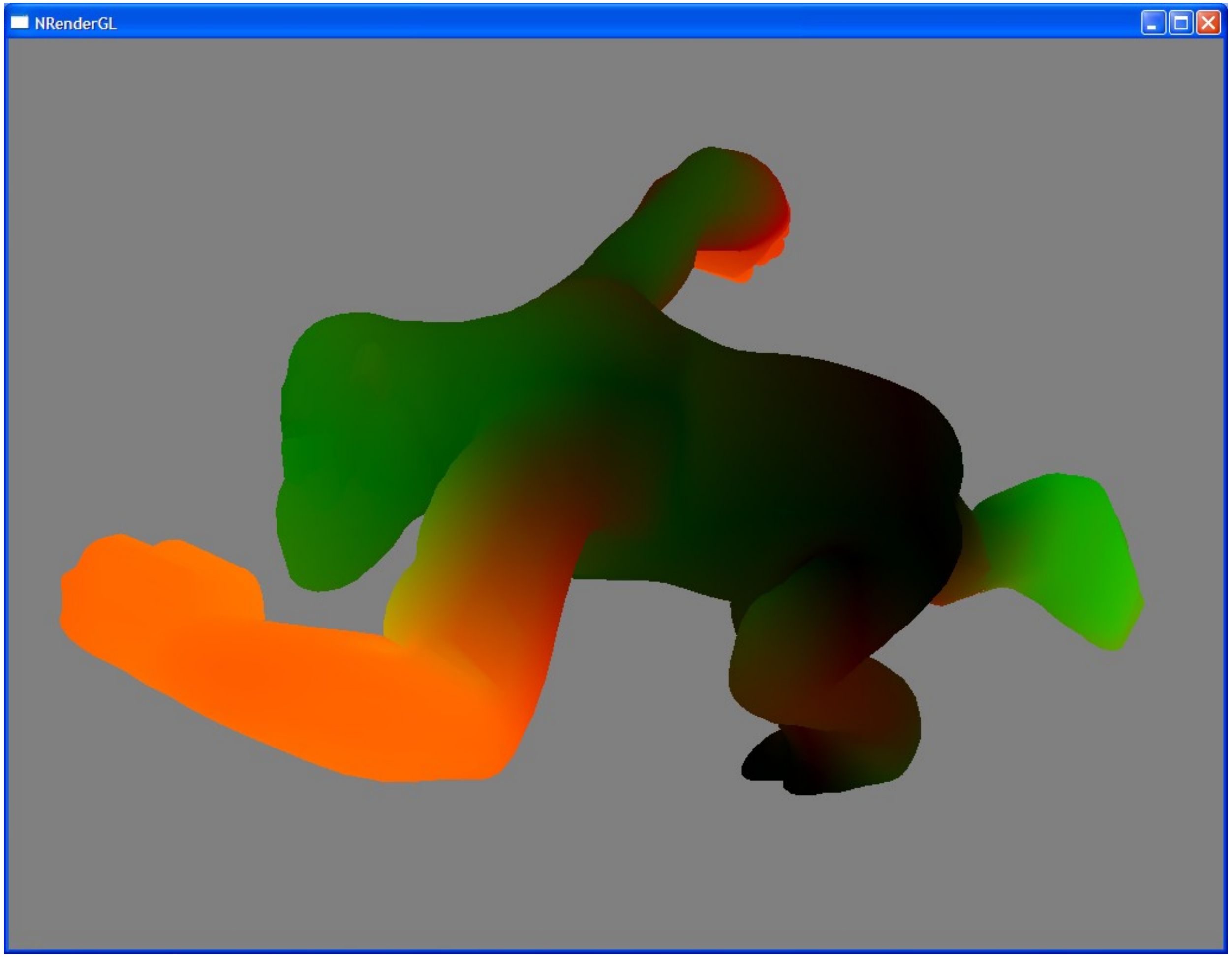
```
struct v2f {
  float4 wpos      : WPOS;
  float3 velocity : TEX0;
};
struct f2f {
  float4 col;
};

f2fConnector main(v2f in,
                  uniform samplerRECT sceneTex,
                  uniform float blurScale = 1.0
                  )
{
  f2f out;
  // read velocity from texture coordinate
  half2 velocity = v2f.velocity.xy * blurScale;

  // sample scene texture along direction of motion
  const float samples = SAMPLES;
  const float w = 1.0 / samples;    // sample weight

  fixed4 a = 0;                     // accumulator
  float i;
  for(i=0; i<samples; i+=1) {
    float t = i / (samples-1);
    a = a + x4texRECT(sceneTex, in.wpos + velocity*t) * w;
  }
  out.col = a;
}
```

# Original Image

# Stretched Geometry



# Velocity Visualization



# Motion Blurred Image

# Future Work

- Stochastic sampling
  - Replaces banding with noise
- Use depth information to avoid occlusion artifacts
- Store image of previous and current frame, interpolate in both directions
- Motion blurred shadows, reflections

# Physical Simulation

- Simple CA-like simulations were possible on previous generation hardware:
  - Greg James' Game of Life / water simulation
  - Mark Harris' CML work
- Use textures to represent physical quantities (e.g. displacement, velocity, force) on a regular grid
- Multiple texture lookups allow access to neighbouring values
- Pixel shader calculates new values, renders results back to texture
- Each rendering pass draws a single quad, calculating next time step in simulation

# Physical Simulation

- Problem: 8 bit precision was not enough, causing drifting, stability problems
- Float precision of new fragment programs allows GPU physics to match CPU accuracy
- New fragment programming model (longer programs, flexible dependent texture reads) allows much more interesting simulations

# Example: Cloth Simulation

- Uses Verlet integration
  - see: Jakobsen, GDC 2001
- Avoids storing explicit velocity
  - new_x = x + (x – old_x)*damping + a*dt*dt
- Not always accurate, but stable!
- Store current and previous position of each particle in 2 RGB float textures
- Fragment program calculates new position, writes result to float buffer / texture
- Then swap current and previous textures

# Cloth Simulation Algorithm

- 4 passes
- Each passes renders a single quad with a fragment program:
  - 1. Perform integration (move particles)
  - 2. Apply constraints:
    - Distance constraints between particles
    - Floor collision constraint
    - Sphere collision constraint
  - 3. Calculate normals from positions using partial differences
  - 4. Render mesh

# Integration Pass Code

```
// Verlet integration step
void Integrate(inout float3 x, float3 oldx, float3 a, float timestep2, float damping)
{
  x = x + damping*(x - oldx) + a*timestep2;
}

fragout_float main(vf30 In,
                   uniform samplerRECT x_tex,
                   uniform samplerRECT ox_tex
                   uniform float timestep = 0.01,
                   uniform float damping = 0.99,
                   uniform float3 gravity = float3(0.0, -1.0, 0.0)
                   )
{
  fragout_float Out;

  float2 s = In.TEX0.xy;

  // get current and previous position
  float3 x =    f3texRECT(x_tex, s);
  float3 oldx = f3texRECT(ox_tex, s);

  // move the particle
  Integrate(x, oldx, gravity, timestep*timestep, damping);

  Out.col.xyz = x;
  return Out;
}
```

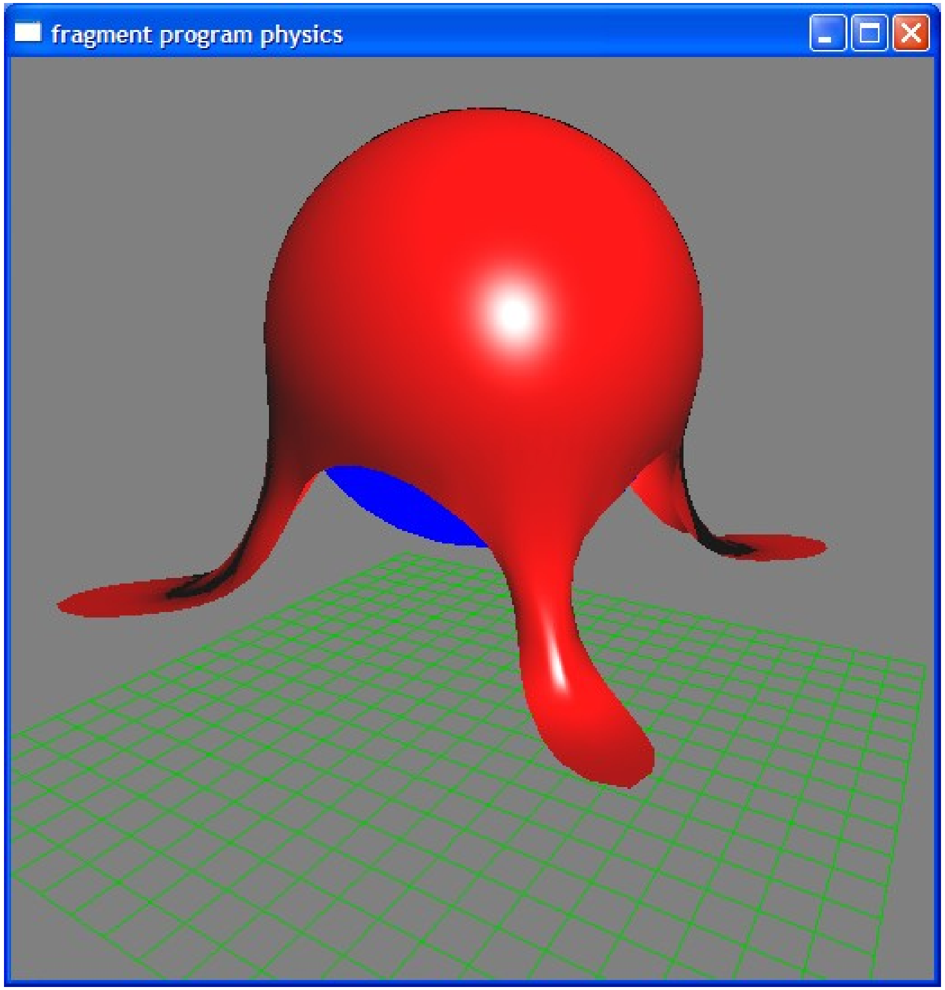
# Constraint Code

```
// constrain a particle to be a fixed distance from another particle
float3 DistanceConstraint(float3 x, float3 x2, float restlength, float stiffness)
{
  float3 delta = x2 - x;
  float deltalength = length(delta);
  float diff = (deltalength - restlength) / deltalength;
  return delta*stiffness*diff;
}

// constrain particle to be outside volume of a sphere
void SphereConstraint(inout float3 x, float3 center, float r)
{
  float3 delta = x - center;
  float dist = length(delta);
  if (dist < r) {
    x = center + delta*(r / dist);
  }
}

// constrain particle to be above floor
void FloorConstraint(inout float3 x, float level)
{
  if (x.y < level) {
    x.y = level;
  }
}
```

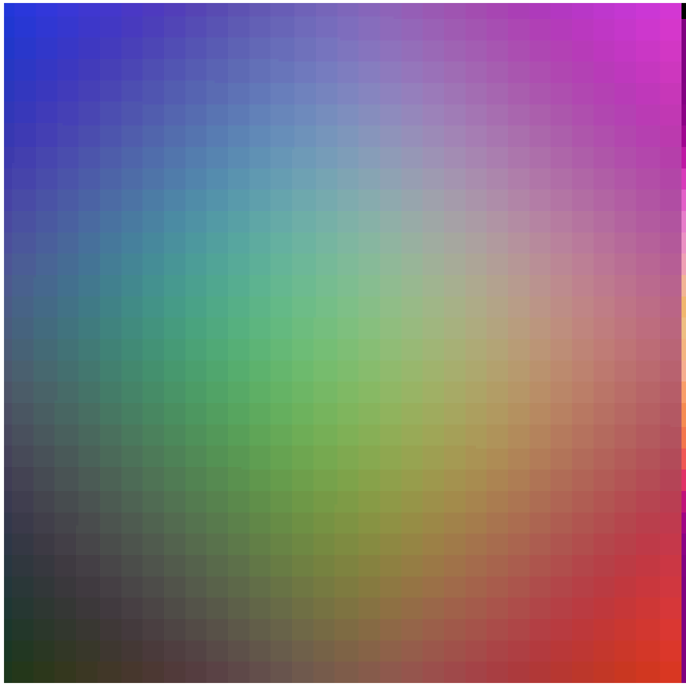
# Constraint Pass Code

```
fragout_float main(vf30 In,
                   uniform texobjRECT x_tex,
                   uniform texobjRECT ox_tex,
                   uniform float meshSize = 32.0,
                   uniform float constraintDist = 1.0,
                   uniform float4 spherePosRad = float3(0.0, 0.0, 0.0, 1.0),
                   uniform float stiffness = 0.2;
                   )
{
  fragout_float Out;
  // get current position
  float3 x = f3texRECT(x_tex, In.TEX0.xy);
  // satisfy constraints
  FloorConstraint(x, 0.0f);
  SphereConstraint(x, spherePosRad.xyz, spherePosRad.z);
  // get positions of neighbouring particles
  float3 x1 = f3texRECT(x_tex, In.TEX0.xy + float2(1.0, 0.0) );
  float3 x2 = f3texRECT(x_tex, In.TEX0.xy + float2(-1.0, 0.0) );
  float3 x3 = f3texRECT(x_tex, In.TEX0.xy + float2(0.0, 1.0) );
  float3 x4 = f3texRECT(x_tex, In.TEX0.xy + float2(0.0, -1.0) );
  // apply distance constraints
  float3 dx = 0;
  if (s.x < meshSize) dx = DistanceConstraint(x, x1, constraintDist, stiffness);
  if (s.x > 0.5)      dx = dx + DistanceConstraint(x, x2, constraintDist, stiffness);
  if (s.y < meshSize) dx = dx + DistanceConstraint(x, x3, constraintDist, stiffness);
  if (s.y > 0.5)      dx = dx + DistanceConstraint(x, x4, constraintDist, stiffness);
  Out.col.xyz = x + dx;
  return Out;
}
```

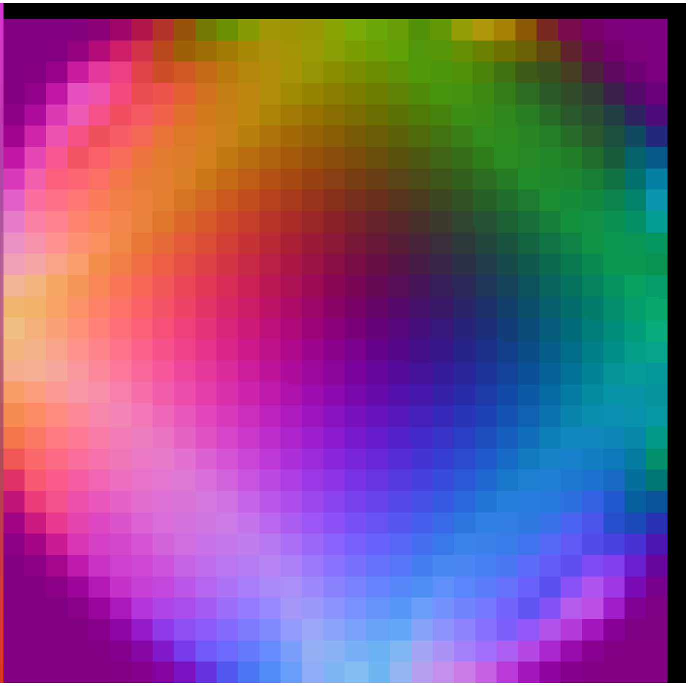
# Screenshot



# Textures



**Position texture**



**Normal texture**

# Render to Vertex Array

- Enables interpretation of floating point textures as geometry
- Possible on NVIDIA hardware using the "NV_pixel_data_range" (PDR) extension
  - Allocate vertex array in video memory (VAR)
  - Setup PDR to point to same video memory
  - Do glReadPixels from float buffer to PDR memory
  - Render vertex array
  - Happens entirely on card, no CPU intervention
- Future ARB extensions may offer same functionality

# Future Work

- Use additional textures to encode particle weights, arbitrary connections between particles (springy objects)
- Collision detection with height fields (encoded in texture)

# References

- ***Advanced Character Physics***, **Thomas Jakobsen, GDC 2001**
- ***A Two-and-a-Half-D Motion-Blur Algorithm,*** **Max and Lerner, Siggraph 1985**
- ***Modeling Motion Blur in Computer-Generated Images,*** **Potmesil, Siggraph 1983**