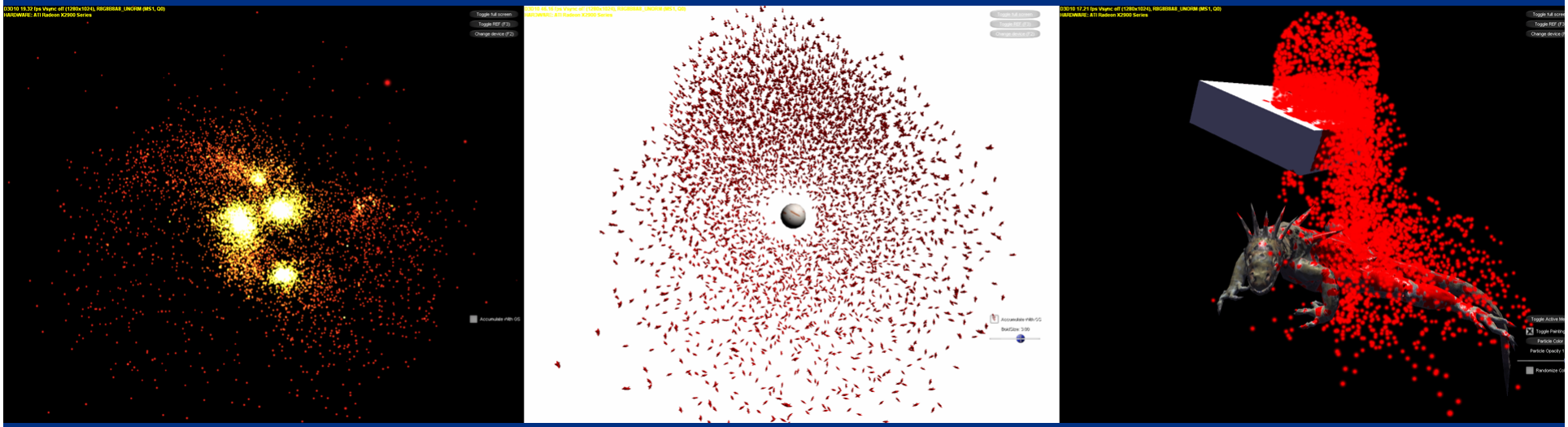
Real-Time Particle Systems on the GPU in Dynamic Environments



Overview

- Introduction
- Rendering system requirements
- Non-parametric GPU particle systems
- Particles that react to other particles
- Particles that react to their environments
- Environments that react to particles

Introduction

- As with everything else, particles systems have made their way to the GPU
- Unfortunately, in doing so, they have lost some of their ability to interact with the environment
- We introduce several methods for bringing interactivity back to GPU particle systems

Rendering system requirements

- Most of these techniques will work on Direct3D 9
- But we will tend towards Direct3D 10
- Specific Direct3D 10 features we will exploit
  - Render to Volume
  - Sampling from buffers at any pipeline stage
  - Stream Out

Non-parametric GPU particle systems

- Storage requirements
- Integrating the equations of motion
- Saving particle states
- Changing behaviors

Non-parametric GPU particle systems: Storage requirements

- We need to store immediate particle state (position, velocity, etc)
- Option 1: Store this data in a vertex buffer
  - Each vertex represents the immediate state of the particle
  - Particles are store linearly in the buffer
- Option 2: Store this data in a floating point texture array
  - First array slice stores positions for all particles.
  - Next array slice stores velocities, etc.

Non-parametric GPU particle systems: Integrating the equations of motion

- Accuracy depends on the length of time step
- We will use Euler integration for these samples
- Runge-Kutta based schemes can be more accurate at the expense of needing to store more particle state

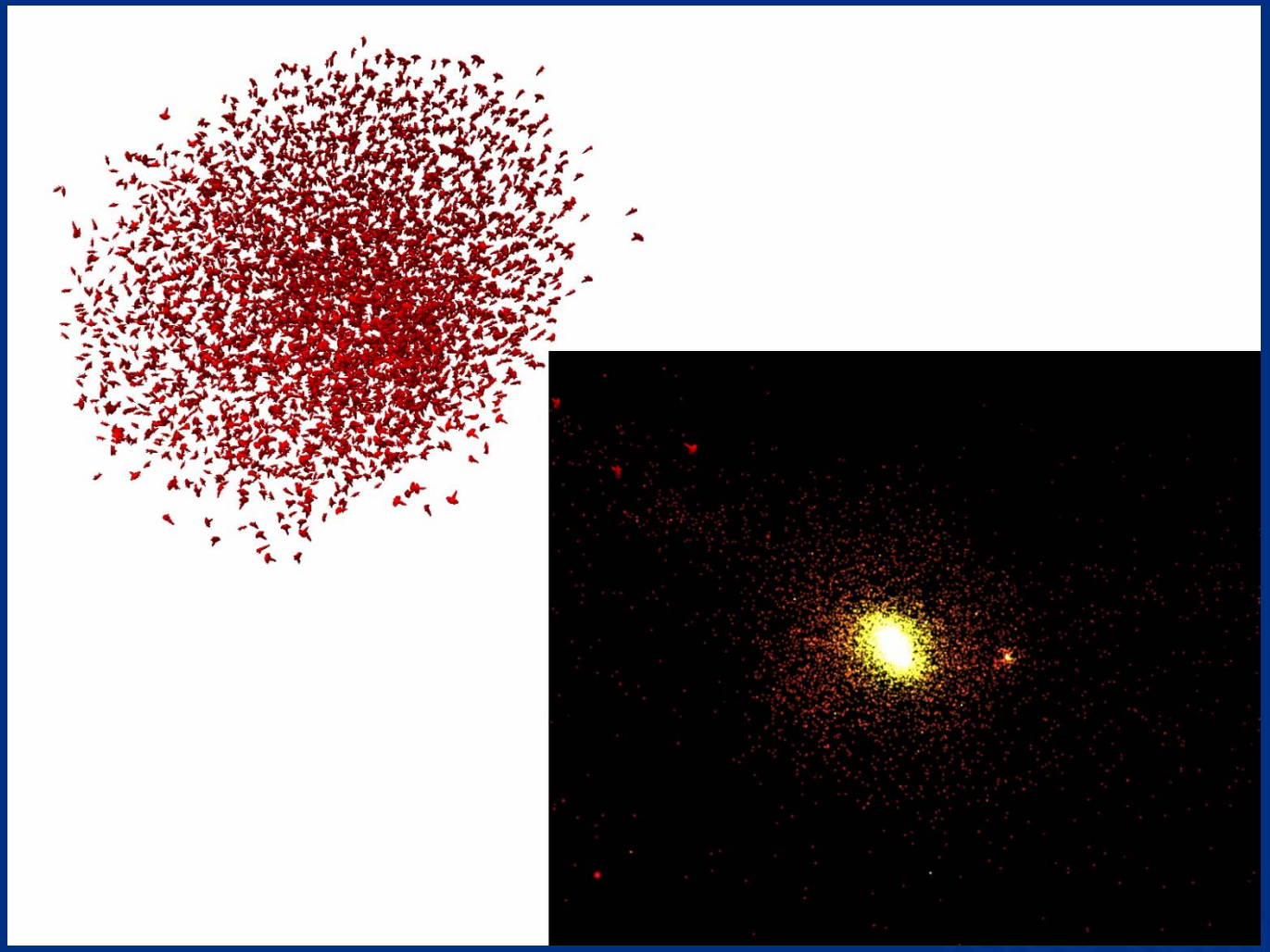
Non-parametric GPU particle systems: Saving particle states

- Integration on the CPU can use read-modify-write operations to update the state in-place
- This is illegal on the GPU (except for non-programming blending hardware)
- Use double buffering
- Ping-pong between them

Non-parametric GPU particle systems: Changing behaviors

- Particles are no longer affixed to a predestined path of motion as in parametric systems.
- Changing an individual velocity or position will change the behavior of that particle.
- This is the basis of every technique in this talk.

Particles that react to other particles



Particles that react to other particles

- N-Body problems
- Force splatting for N^2 interactions
- Gravity simulation
- Flocking particles on the GPU

Particles that react to other particles: N-Body problems

- Every part of the system has the possibility of affecting every other part of the system
- Space partitioning can help
- Treating multiple parts of a system at a distance as one individual part can also help
- Parallelization can also help (GPU)

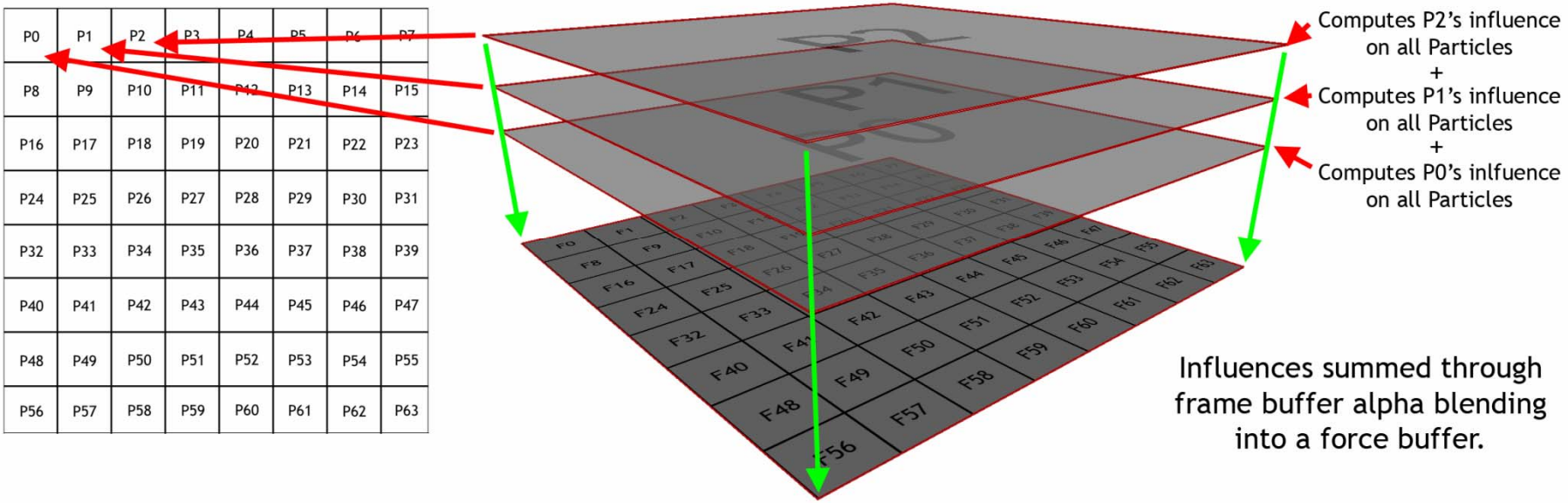
Particles that react to other particles: Force splatting for N^2 interactions

- Our goal is to project the force from one particle onto all other particles in the system
- Create a texture buffer large enough to hold forces for all particles in the simulation
- Each texel holds the accumulated forces acting upon a single particle

Particles that react to other particles: Force splatting for N^2 interactions

- Create a stack of N quads (N is the number of particles)
- VS: For each quad 1..N, sample particle N's state from the buffer or texture
- PS: Sample the target particle based on our interpolated UV
- Determine the force from the VS particle to the PS particle and output it

Particles that react to other particles: Force splatting for N^2 interactions



Particles that react to other particles: Force splatting for N^2 interactions

```
float4 PSAccumulateForce(PSForceIn input) : SV_Target
{
    float3 inplacePos = g_txParticleData.SampleLevel( g_samPoint,
                                                      float3(input.tex, 0),
                                                      0 );

    // use the law of gravitation to calculate the force of the input
    // particle on the particle we're rasterizing across at this moment
    float3 delta = input.pos - inplacePos;
    float r2 = dot( delta, delta );

    float4 force = float4(0,0,0,0);
    if( r2 > 0 )
    {
        //Calculate force of the input object on the inplace object
        …
    }

    return force;
}
```

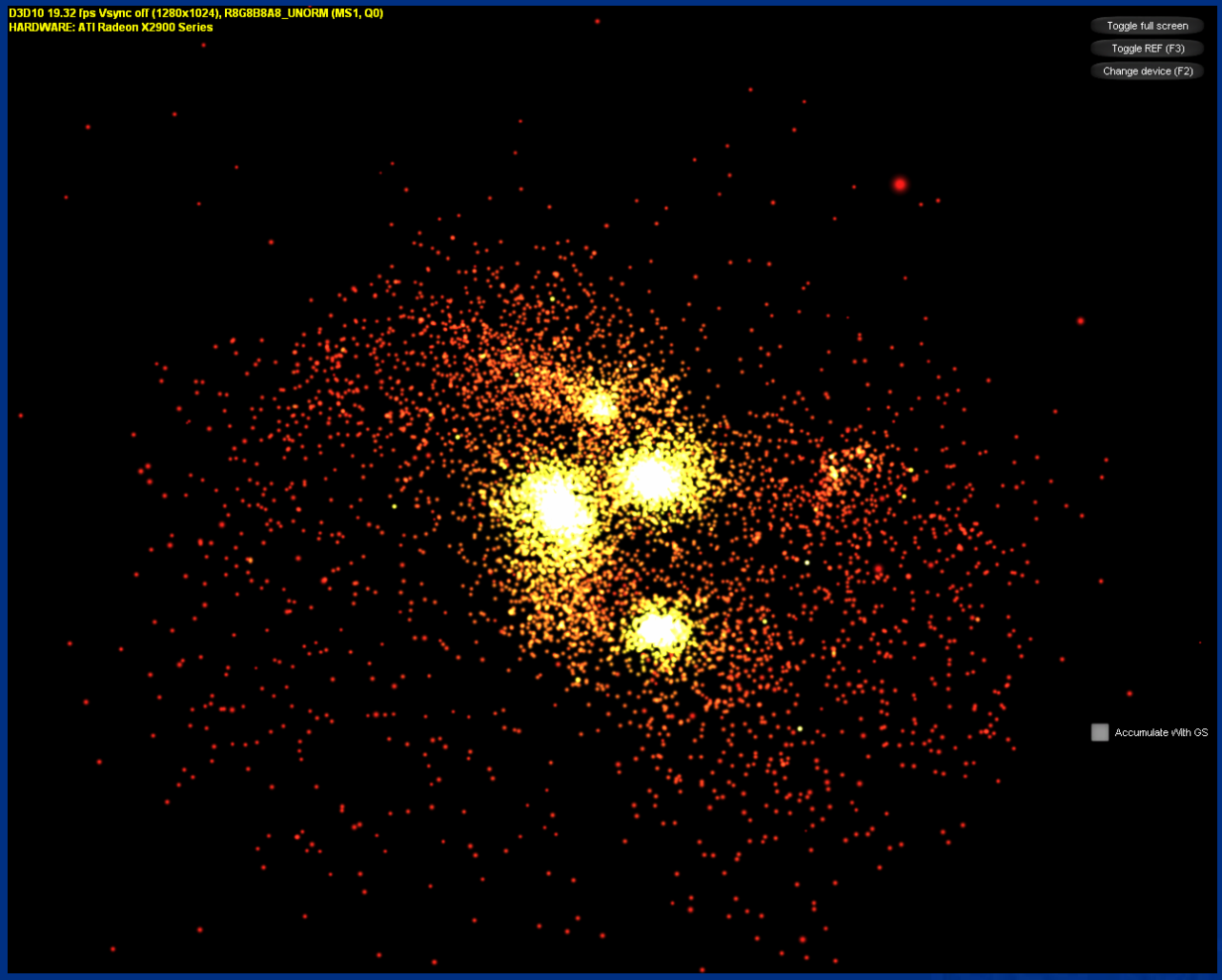
Sample the particle that we will be applying the force to.

Our input particle data comes into the shader via the interpolators.

Particles that react to other particles: Force splatting for N^2 interactions

- Yes, it's $O(N^2)$, but it exploits the fast rasterization, blending, and SIMD hardware of the GPU
- It also avoids the need to create any space partitioning structures on the GPU

Particles that react to other particles: Gravity simulation



Particles that react to other particles: Gravity simulation

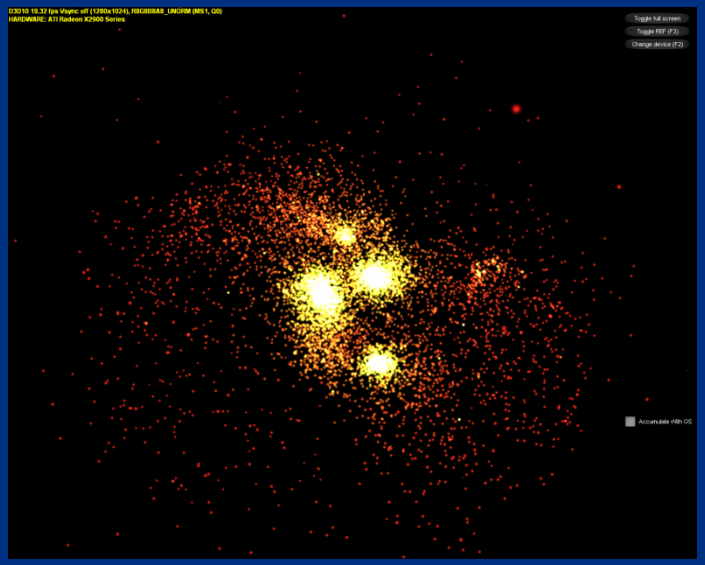
- Uses force splatting to project the gravitational attraction of each particle onto every other particle
- Handled on the GPU
  - CPU only sends time-step information to the GPU
  - CPU also handles high-level switching of render targets and issuing draw calls

Particles that react to other particles: Gravity simulation

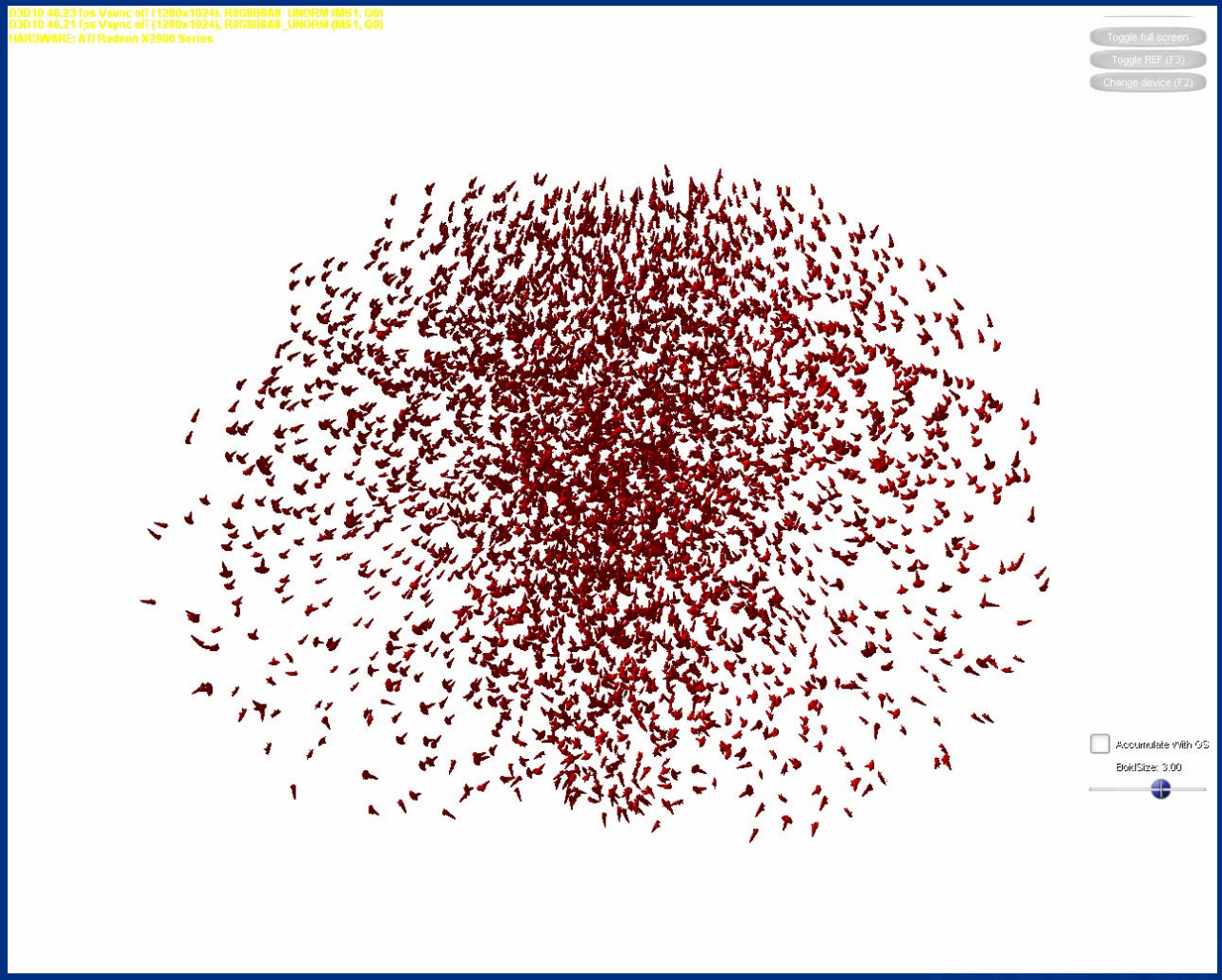
```
float4 PSAccumulateForce(PSForceIn input) : SV_Target
{
    float3 inplacePos = g_txParticleData.SampleLevel( g_samPoint,
                                                      float3(input.tex, 0),
                                                      0 );

    // use the law of gravitation to calculate the force of the input
    // particle on the particle we're rasterizing across at this moment
    float3 delta = input.pos - inplacePos;
    float r2 = dot( delta, delta );

    float4 force = float4(0,0,0,0);
    if( r2 > 0 )
    {
        // Calculate acceleration between the two caused by gravity
        float r = sqrt(r2);
        float3 dir = delta/r;
        force.xyz = dir * g_fG * ( (g_fParticleMass * g_fParticleMass) / r2 );
    }

    return force;
}
```

Calculate attraction between two point masses.

Demo: Gravity Simulation



Particles that react to other particles: Flocking particles on the GPU



Particles that react to other particles: Flocking particles on the GPU

- Uses basic boids behaviors [Reynolds87,Reynolds99] to simulation thousands of flocking spaceships on the GPU
  - Avoidance
  - Separation
  - Cohesion
  - Alignment

Particles that react to other particles: Flocking particles on the GPU

- Avoidance and Separation
- If a collision between two ships is imminent, a force vector is applied that steers the ships away from the collision
- If a comfortable distance between two ships is in jeopardy, a force is applied to spread the ships apart

Particles that react to other particles: Flocking particles on the GPU

```
float4 PSAccumulateForce(PSForceIn input) : SV_Target
{
    float3 texcoord = float3( input.tex, 0 );
    float3 vLocalPos = g_txParticleData.SampleLevel( g_samPoint, texcoord, 0 );
    texcoord.z = 1;
    float3 vLocalVel = g_txParticleData.SampleLevel( g_samPoint, texcoord, 0 );

    // Do we need to do anything?
    float3 vNeighborPos = input.pos;
    float3 delta = vLocalPos - vNeighborPos;
    float r2 = dot( delta, delta );
    float3 vTotalForce = float3(0,0,0);
    if( r2 > 0 )
    {
        vTotalForce = Separation( vLocalPos, vNeighborPos );
        vTotalForce += Avoid( vLocalPos, vLocalVel, vNeighborPos );
    }

    return float4(vTotalForce,1);
}
```

Any interactions between bodies goes here.

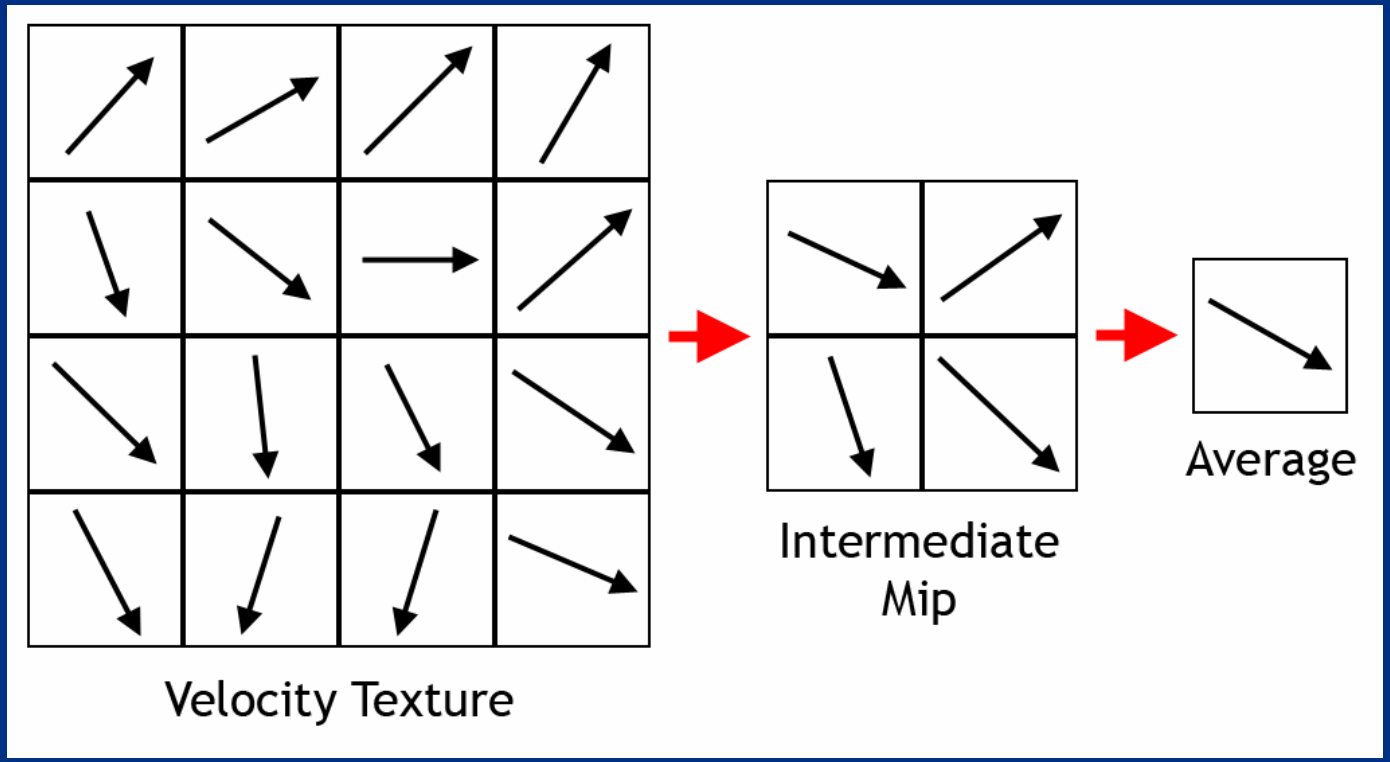
Particles that react to other particles: Flocking particles on the GPU

- Cohesion and Alignment
- Unlike N-Body problems, Cohesion and Alignment need the average of all particle states
  - Cohesion steers ships toward the common center.
  - Alignment steers ships toward the common velocity
- We could average all of the positions and velocities by repeatedly down-sampling the particle state buffer / texture
- However, the graphics system can do this for us

Particles that react to other particles: Flocking particles on the GPU

- Graphics infrastructure can already perform this quick down-sampling by generating mip maps
- The smallest mip in the chain is the average of all values in the original texture

Particles that react to other particles: Flocking particles on the GPU



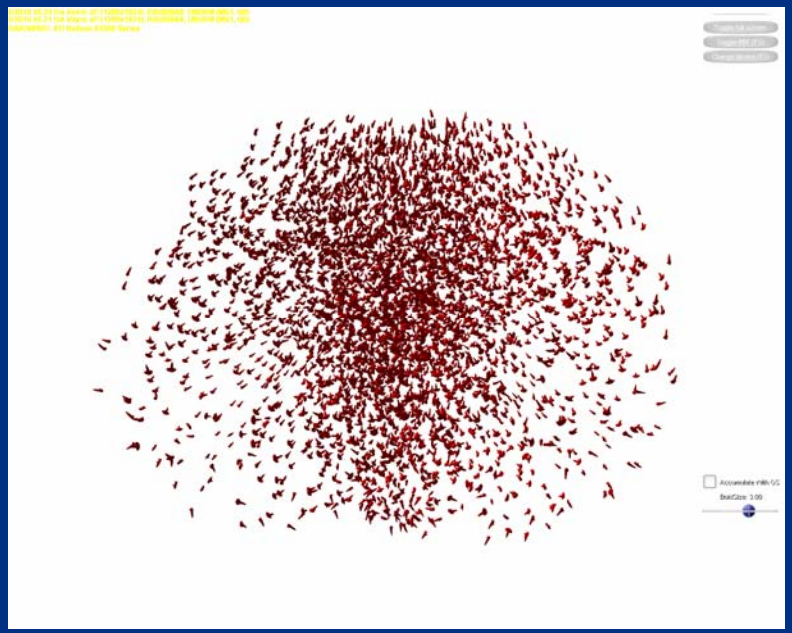
Particles that react to other particles: Flocking particles on the GPU

```
// Our texcoord is interpolated from the full-screen quad
float3 texcoord = float3( input.tex, 0 );

// Sample the current position in the position texture
float3 pos = g_txParticleData.SampleLevel( g_samPoint, texcoord, 0 );

// Sampling the average position is as easy as sampling any point from the
// smallest mip map in the position texture.
float3 avgPos = g_txParticleData.SampleLevel( g_samPoint, texcoord, g_iMaxMip );

// Sampling the average velocity is as easy as sampling any point from the
// smallest mip map in the velocity texture.
texcoord.z = 1;
float4 avgVel = g_txParticleData.SampleLevel( g_samPoint, texcoord, g_iMaxMip );
```

We store position and velocity as two slices of a texture array. The z-coordinate determines which slice to sample from.

Remember, average position and average velocity are in the smallest mip map.

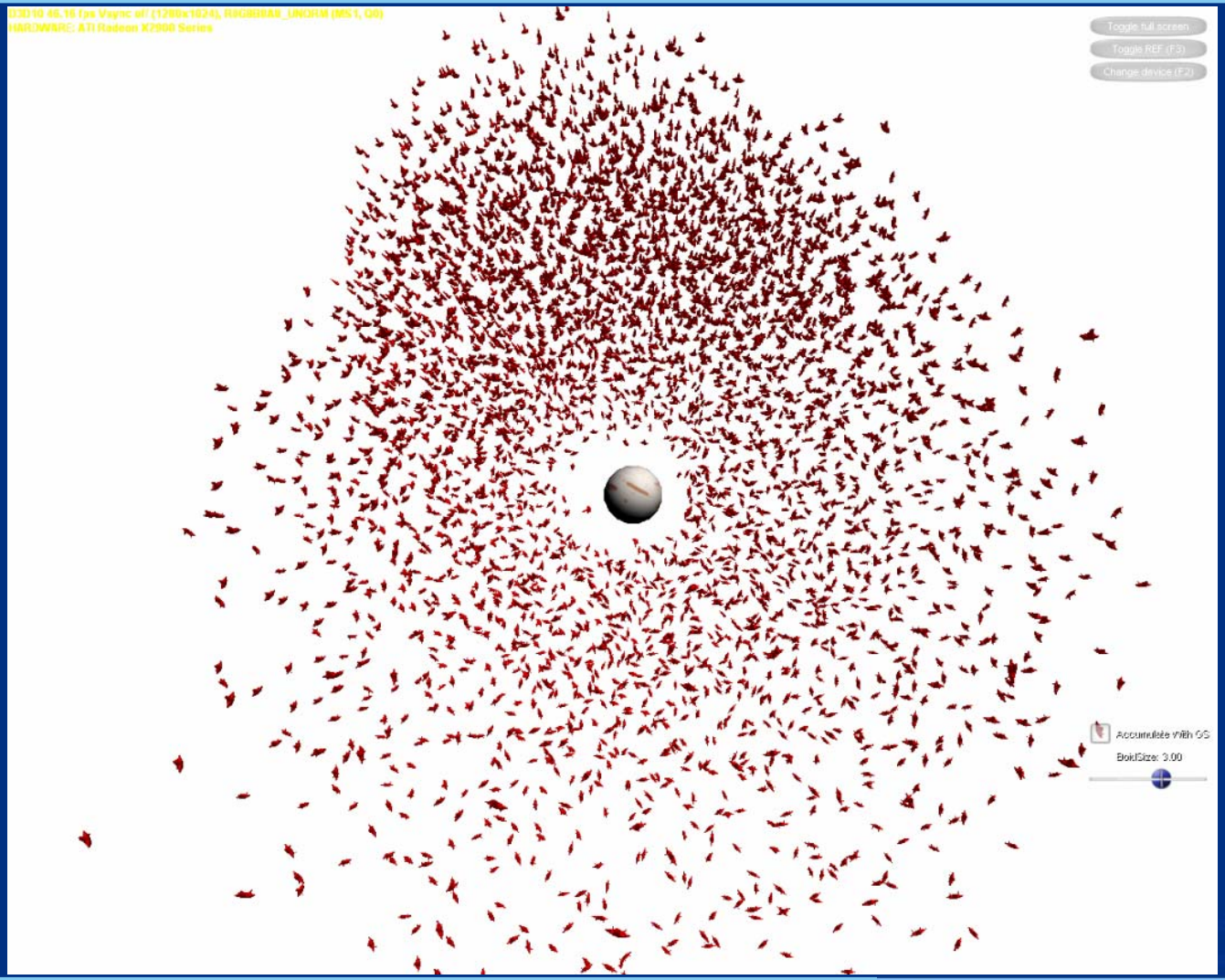
Demo: Flocking Spaceships



Particles that react to their environments

- Reacting to spherical objects
- Reacting to arbitrary objects using render to volume

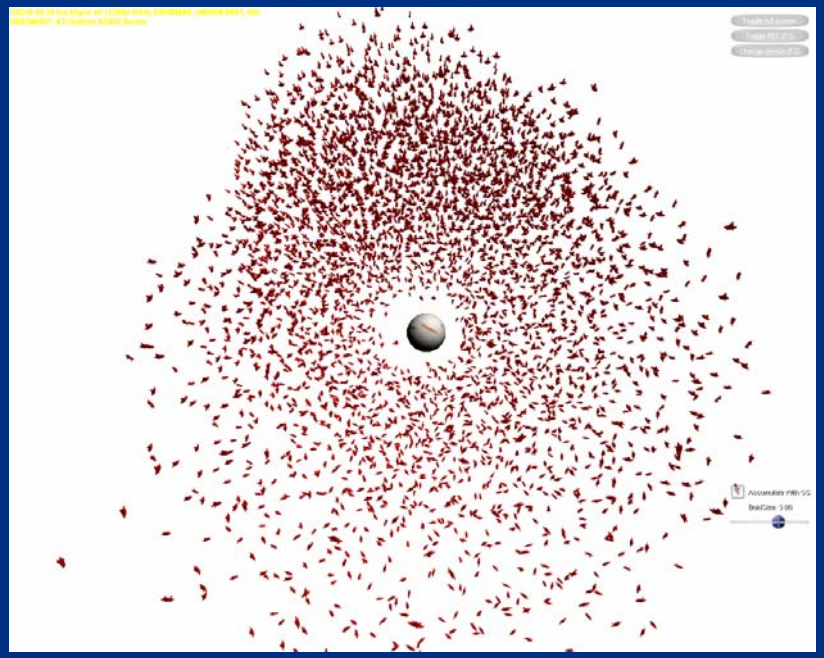
Particles that react to their environments: Reacting to spherical objects



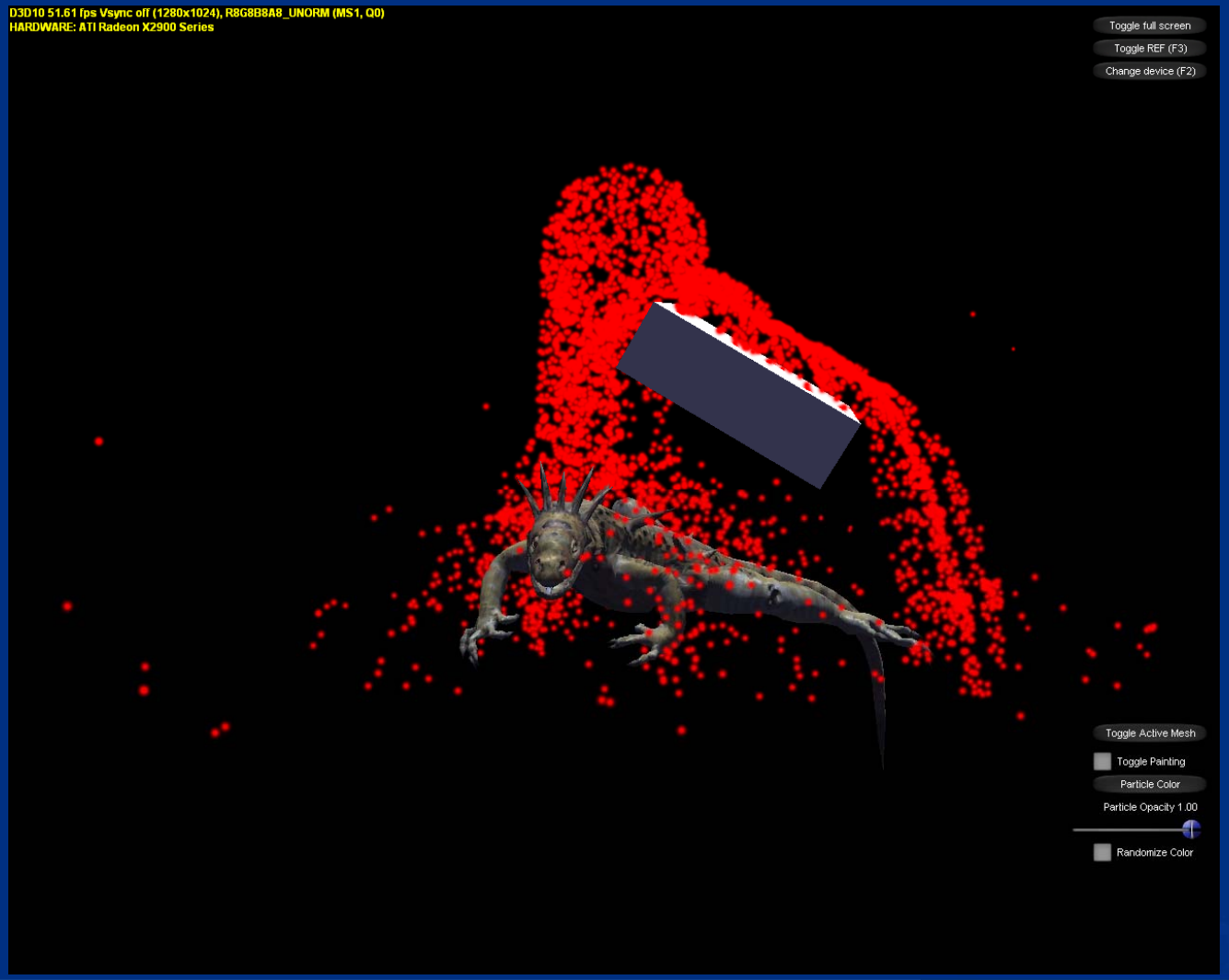
Particles that react to their environments: Reacting to spherical objects

- A very simple way to interact with a particle system is to influence it through a limited set of “point charges”
- We can repel or attract an entire flock by sampling point charges
- Point charge corresponds to the location and size of an object in the scene

Demo: Reacting to Point Charges



Particles that react to their environments: Reacting to arbitrary objects using RTV



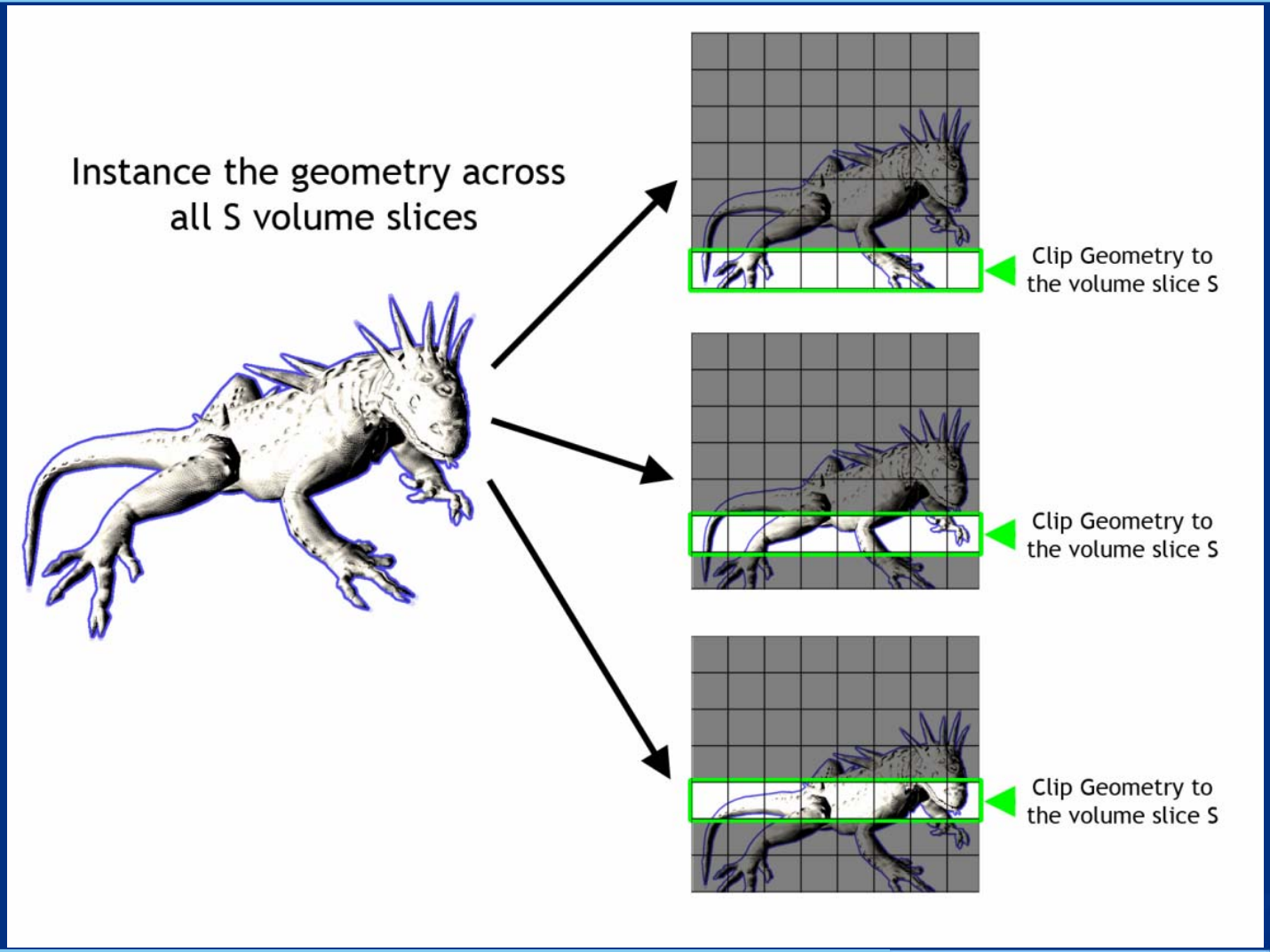
Particles that react to their environments: Reacting to arbitrary objects using RTV

- Point charges are nice, but not every object can be approximated by a set of spheres
- We would like to interact with more complex objects, like skinned characters
- Previous approaches partitioned space into a 2D grid [Lutz04]
- This limited the problem to interaction with a heightfield

Particles that react to their environments: Reacting to arbitrary objects using RTV

- We can extend this to 3 dimensions by rendering the scene into a volume texture
- We instance the geometry across all slices
- For each instance we clip the geometry to the slice
- For each voxel in the volume texture stores the plane equation and velocity of the scene primitive.

Particles that react to their environments: Reacting to arbitrary objects using RTV



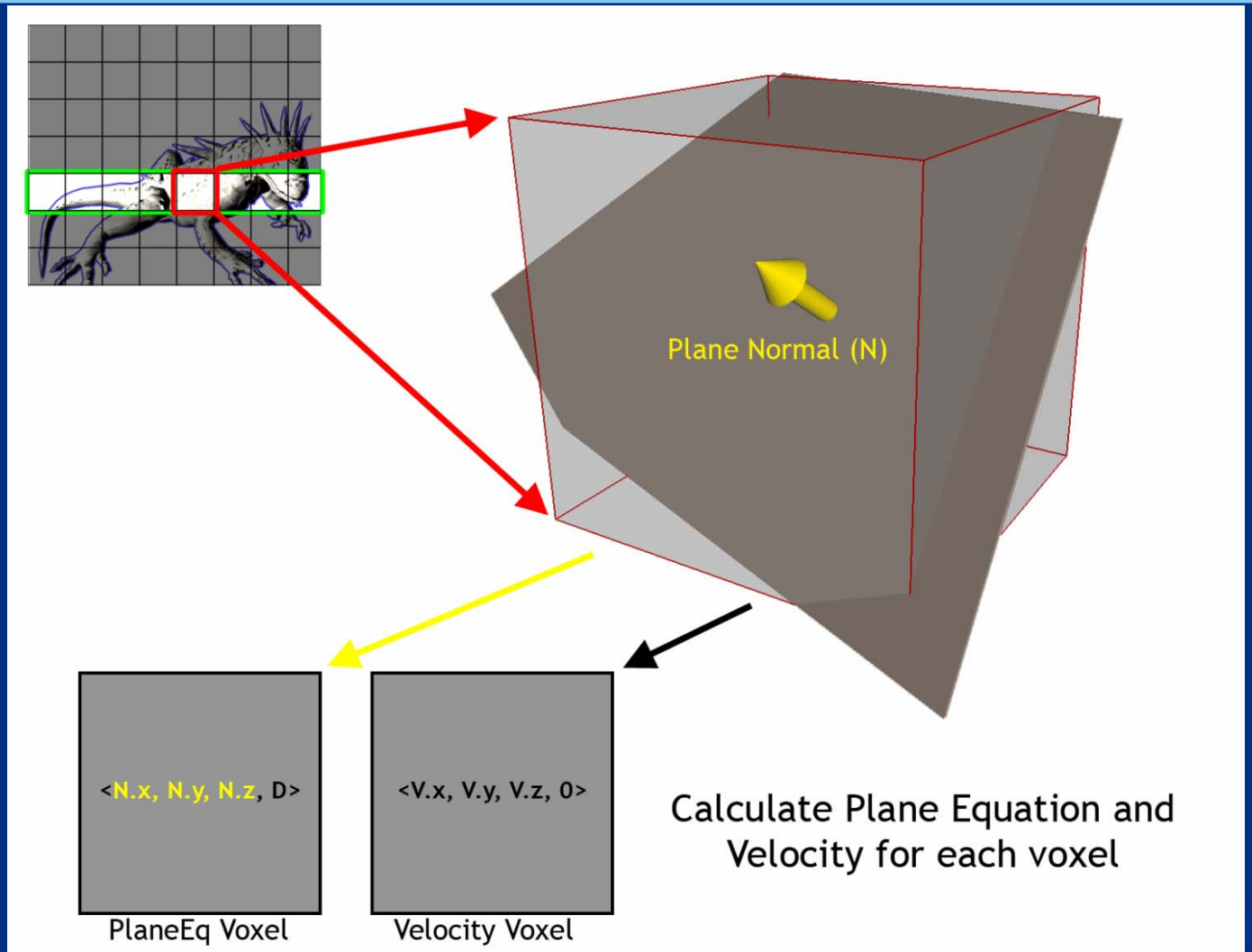
Particles that react to their environments: Reacting to arbitrary objects using RTV

```
struct GSSceneOut
{
    float4 PlaneEq          : NORMAL;
    float2 PlaneDist        : CLIPDISTANCE;
    float4 Pos              : SV_POSITION;
    uint   RTIndex          : SV_RENDERTARGETARRAYINDEX;
};

[maxvertexcount(3)]
void GSScene( triangle GSSceneIn input[3], inout TriangleStream<GSSceneOut> TriStream )
{
        GSSceneOut output;
        …
        // send us to the right render target
        output.RTIndex = input[0].InstanceID;
        …
}
```

Think of the volume render target as a giant texture array.

We can specify the volume slice to render into by specifying the RTIndex

Particles that react to their environments: Reacting to arbitrary objects using RTV



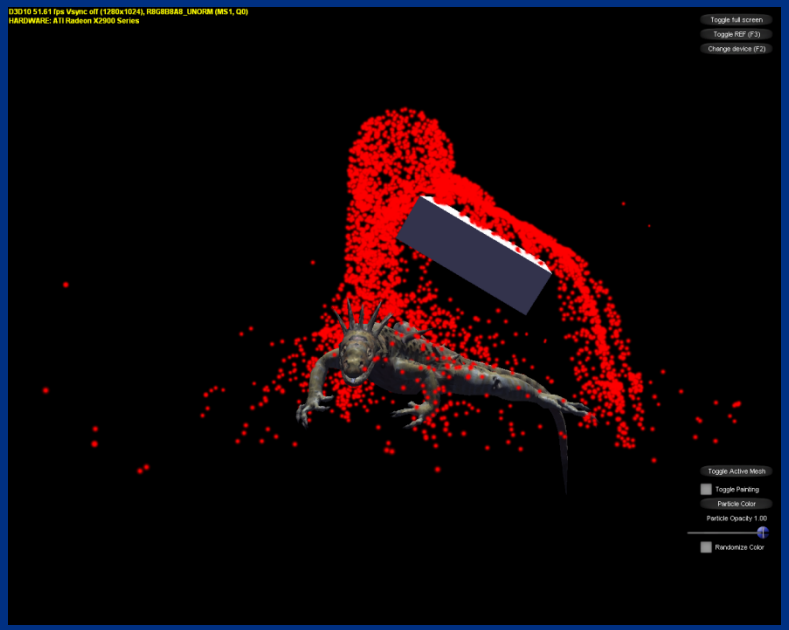
Particles that react to their environments: Reacting to arbitrary objects using RTV

```
[maxvertexcount(3)]
void GSScene( triangle GSSceneIn input[3], inout TriangleStream<GSSceneOut> TriStream )
{
        GSSceneOut output;

        // calculate the face normal
        float3 faceEdgeA = input[1].wPos - input[0].wPos;
        float3 faceEdgeB = input[2].wPos - input[0].wPos;
        float3 faceNormal = normalize( cross( faceEdgeA, faceEdgeB ) );

        // find the plane equation
        float4 planeEq;
        planeEq.xyz = faceNormal;
        planeEq.w = -dot( input[0].wPos, faceNormal );

        …
        // Output the triangle
        …
}
```

Particles that react to their environments: Reacting to arbitrary objects using RTV

- For each particle, we determine the voxel it lives in
- We sample the voxel in the shader and collide the particle with volume using the stored plane equation and velocity

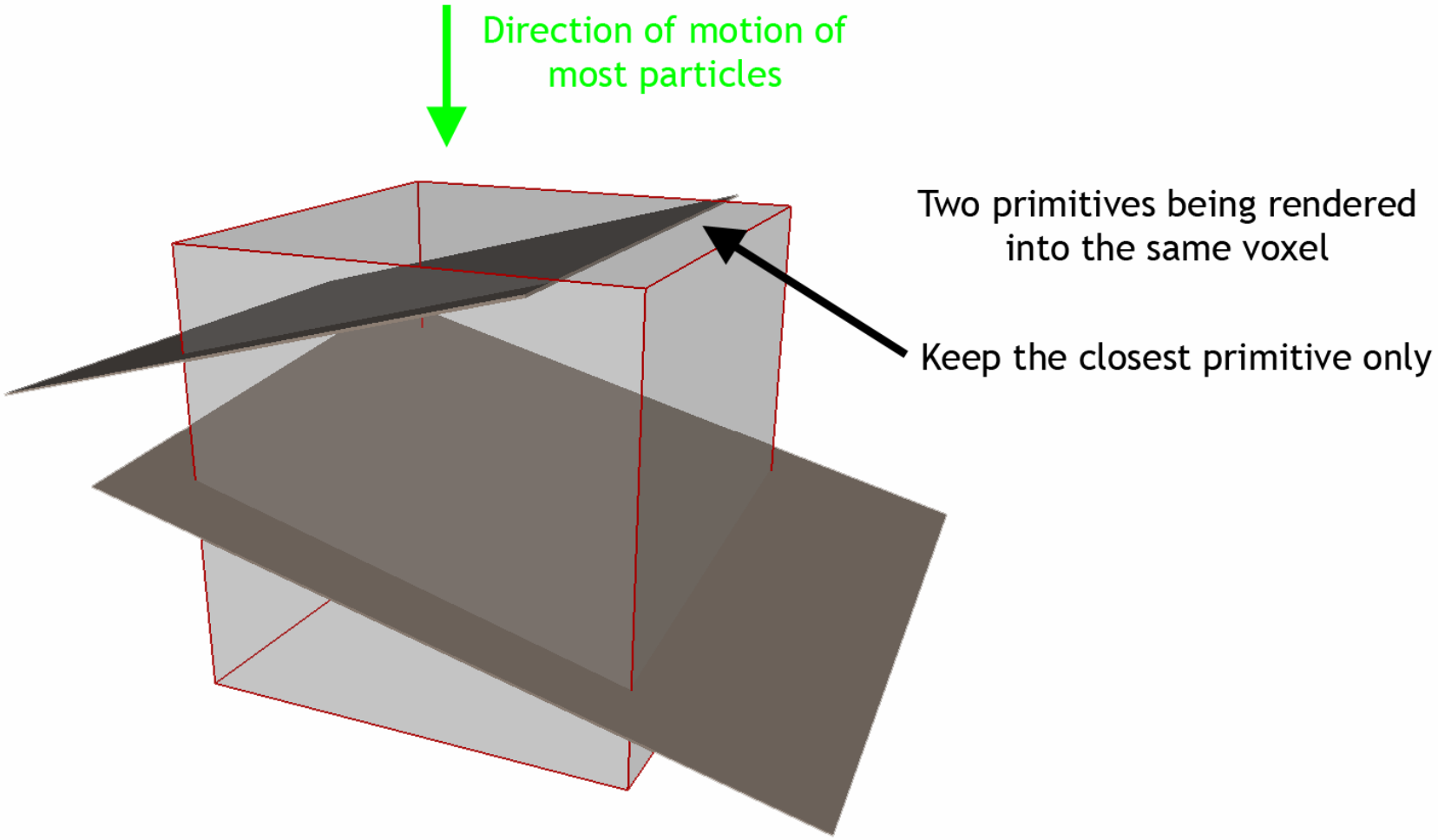
Demo: Particles and the Lizard



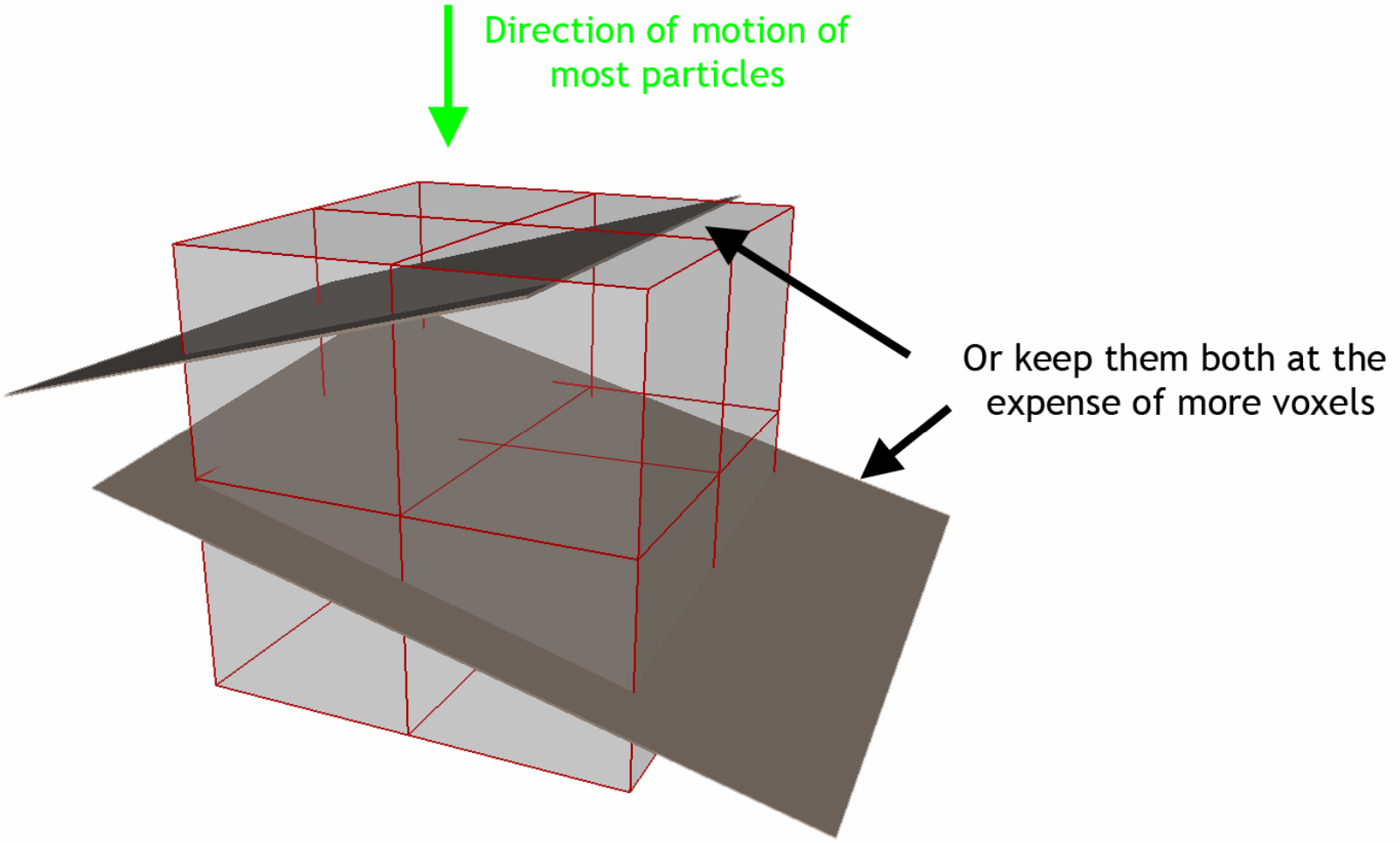
Particles that react to their environments: Reacting to arbitrary objects using RTV

- With detailed geometry or a coarse grid, multiple primitives may be rasterized into the same voxel
- To reduce the aliasing we can always increase the size of the volume texture at the expense of memory usage
- Another approach is to only store the primitive that is closest to the average velocity of the system
  - This ensures that the majority of particles interact correctly.
  - This also degenerates to the height-field case where two primitives alias.

Particles that react to their environments: Reacting to arbitrary objects using RTV



Particles that react to their environments: Reacting to arbitrary objects using RTV



Environments that react to particles

Environments that react to particles

- Painting with particles using a gather approach

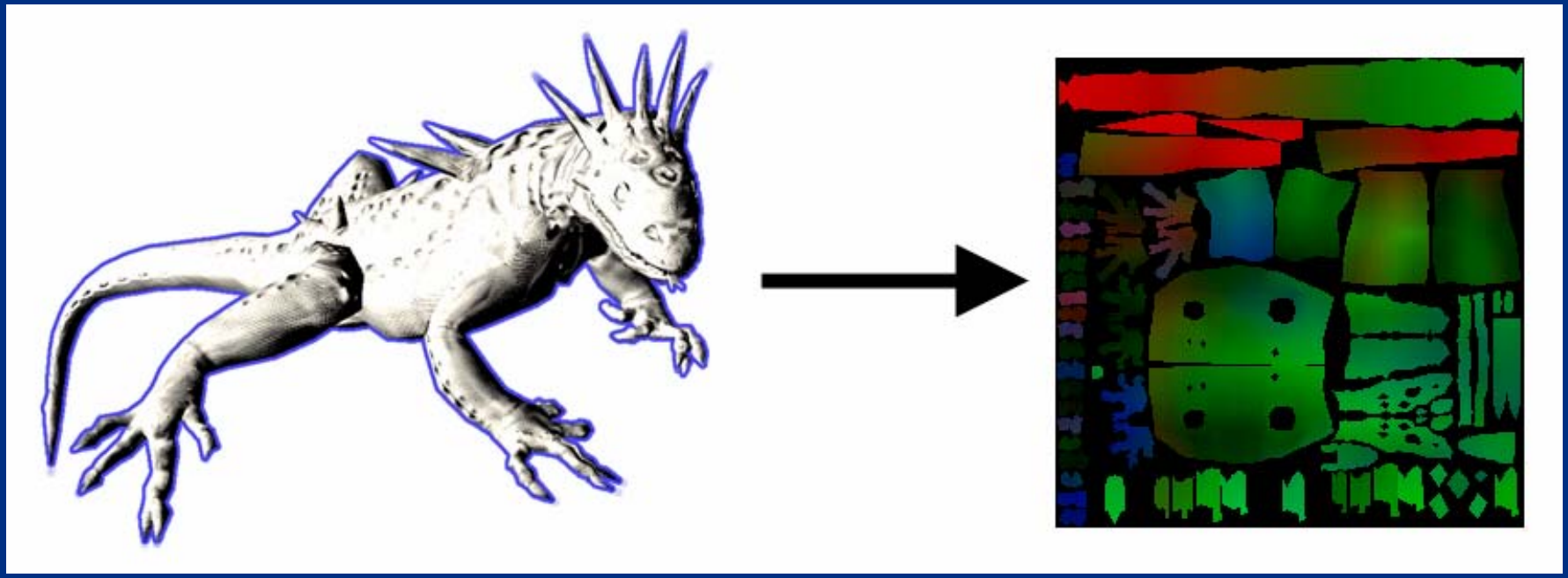
Environments that react to particles: Painting with particles using gather

- We also want particles to affect the environment
- One example of this is painting with particles
- When a particle hits a mesh in the scene, it imparts its color onto that mesh

Environments that react to particles: Painting with particles using gather

- First we need a representation of the position of the mesh primitives in texture space
- We will call this the position buffer
- We create this by rendering the mesh into a texture using the UV coordinate as the output clip coordinate and the transformed position of the mesh in world space as the color
- In order to give correct results, this requires a unique mapping of the mesh

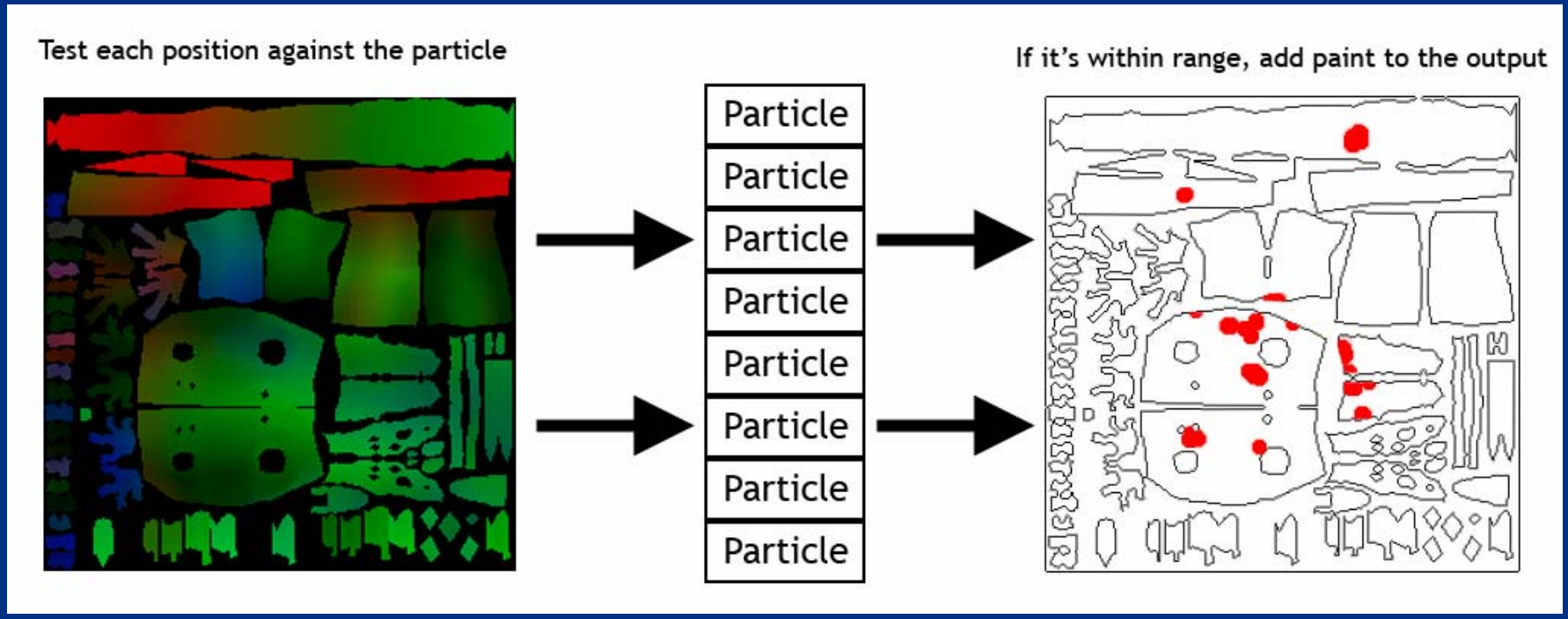
Environments that react to particles: Painting with particles using gather



Environments that react to particles: Painting with particles using gather

- Once we have the position buffer, we can gather paint splotches.
- We use a gather pixel shader to traverse the position buffer.
- For each particle in the buffer, the shader samples its position and determines if it can affect the current position in the position buffer.
- If it can, the color of the particle is sent to the output render target, which doubles as the mesh texture.

Environments that react to particles: Painting with particles using gather



Particles that react to their environments: Painting with particles using gather

```
float4 PSPaint(PSQuadIn input) : SV_Target
{
    // sample the position buffer
    float4 meshPos = g_txDiffuse.Sample( g_samPoint, input.Tex );

    // loop through the particles
    float3 color = float3(0,0,0); float alpha = 0;
    for( int i=g_ParticleStart; i<g_NumParticles; i+=g_ParticleStep )
    {
            // load the particle data from the buffer
            float4 particlePos = g_ParticleBuffer.Load( i*4 );
            float4 particleColor = g_ParticleBuffer.Load( (i*4) + 2 );

            float3 delta = particlePos.xyz - meshPos.xyz;
            float distSq = dot( delta, delta );
            if( distSq < g_fParticleRadiusSq )
            {
                color = color.xyz*(1-particleColor.a) + particleColor.xyz * particleColor.a;
                alpha += particleColor.a;
            }
    }

    return saturate( float4(color,alpha) );
}
```

Load the particle data directly from the particle buffer.

Demo: Painting with Particles

Environments that react to particles: Painting with particles using gather

- With large numbers of particles, the test bewteen each position in the position buffer and each particle in the particle buffer becomes expensive
- To mitigate this, we can amortize the cost over time.
  - In this case, we only test a certain percentage of the particles during each frame
  - EG. We can test the first 1/5th of the particles for the first frame, the next 1/5th of the particles the next frame, etc

Acknowledgements

- Thanks to Matt Dudley for the animated lizard mesh

References

- [Blythe06] Blythe, D. 2006. The Direct3D 10 system. ACM Trans. Graph. 25, 3, 724-734.
- [Burg00] J. van der Burg. Building an Advanced Particle System. Gamasutra, June 2000.
- [Lutz04] Lutz, L. (2004). Buildling a Million Particle System. In the proceedings of Game Developers Conference, San Francisco, CA, March 2004.
- [McAllister00] David K. McAllister. The design of an API for Particles Systems. University of North Caroline Technical Report TR 00-007.
- [Reeves83] W. T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. ACM Transactions on Graphics, 2(2):91—108, Apr. 1983.

References

- [Reynolds87] C. Reynolds. Flocks, Herds and Schools: A Distributed Behavioural Model. Computer Graphics, 21(4), 1987, pp.25—34.
- [Reynolds99] Reynolds, C. W. (1999) Steering Behaviors for Autonomous Characters, in the proceedings of Game Developers Conference 1999 held in San Jose, California. Miller Freeman Game Group, San Francisco, California. pp. 763-782.
- [Sims90] K. Sims. Particle Animation and Rendering Using Data Parallel Computation. ACM Computer Graphics (SIGGRAPH '90), 24(4):405--413, August 1990.