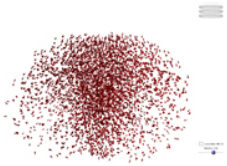
GPUBoids Sample

Collapse All

This is one of 3 sample applications shown during the Advanced Real-Time Rendering in 3D Graphics and Games course at SIGGraph 2007. The Direct3D 10 sample shows a flocking algorithm managed entirely by the GPU.



Path

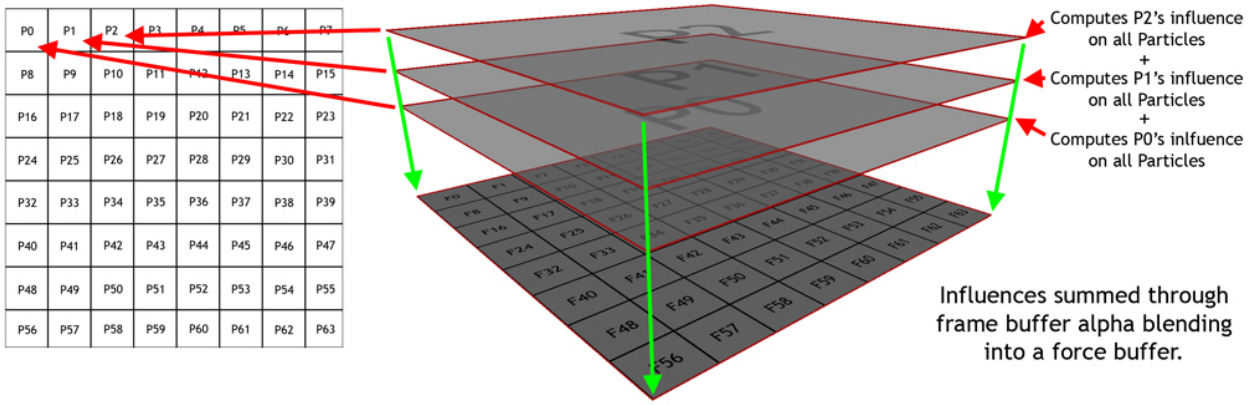
Source	*SDK root*\Samples\C++\Direct3D10\GPUBoids
Executable	*SDK root*\Samples\C++\Direct3D10\Bin*x86 or x64*\GPUBoids.exe

How the Sample Works

Oftentimes particle systems are used to create the illusion of flocks of birds or bugs swarming around a light or fallen comrade. Traditional flocking behaviors need to follow a few simple rules in order to look plausible. In this situation, the rules are collision avoidance, separation, cohesion, and alignment. See [Reynolds87, Reynolds99] for an in-depth descriptions of flocking behaviors.

Force Splatting

The goal of force splatting is to project the force from one particle onto all other particles during a single operation. In this case, the operation is the rendering of a quad primitive. We create a texture that acts and an accumulation buffer for all forces applied to the particles. This buffer will be the target of the rasterization operations that will accumulate particle forces. Each texel in the force texture holds the accumulated forces acting upon a single particle. We also create a stack of N quad primitives, where N is the number of particles in the system. The dimensions of dimensions of the quads are such that they will exactly cover the force buffer when rasterized. The four vertices of each quad in the stack contain a vertex element which identifies the exact particle represented by the quad. During rasterization, this interpolated vertex element is used to fetch properties of the particle from the particle texture or the particle buffer.



During the rasterization of a single quad, the forces are calculated between the particle being rasterized to and the particle represented by the vertex element in the vertices of the quad. Forces are accumulated by rendering successive quad with additive alpha blending enabled.

While inelegant less than elegant in terms of algorithmic complexity, the force splatting algorithm exploits the fast rasterization and alpha blending capabilities of modern graphics hardware without the need to continually recreate complex space partitioning structures on the GPU.

Separation and Avoidance

The flocking simulation takes advantage of the previous N*N force splatting to avoid collisions between particles as well as to maintain a certain comfortable separation between all particles. Instead of computing the gravitational attraction between particles (as in the NBodyGravity sample), we compute a repellant force for each particle based upon either how close the particles are to colliding or how much space is between particles.

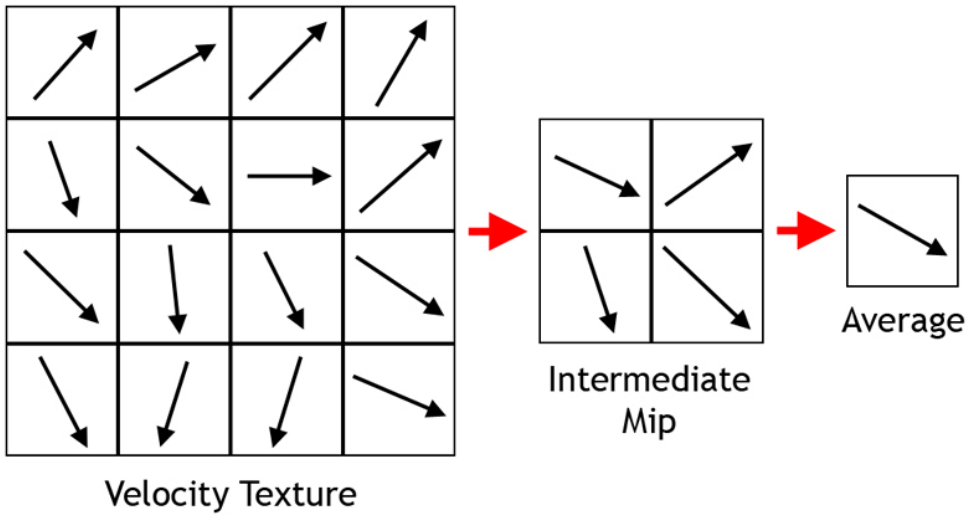
```
float4 PSAccumulateForce(PSForceIn input) : SV_Target
{
    float3 texcoord = float3( input.tex, 0 );
    float3 vLocalPos = g_txParticleData.SampleLevel( g_samPoint, texcoord, 0 );
    texcoord.z = 1;
    float3 vLocalVel = g_txParticleData.SampleLevel( g_samPoint, texcoord, 0 );

    // Check neighbors for separation and avoidance
    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);
}
```

Fast Averaging for Cohesion and Alignment

Behaviors such as cohesion and alignment rely on the knowledge of the average position and average velocity of the particles respectively. Fortunately, modern graphics hardware provides a fast way of averaging entire textures by being able to generate mip-maps on the fly. By sampling from the smallest mip-level during the particle update phase, we can create a force vector from the particle to the center of mass for cohesion or create a force vector that aligns our particle with the average velocity of all other particles. This force vector is added to the force vector sampled from the force accumulation texture.



The following snippet shows how to sample the average position and velocity from the smallest mip in the texture.

```
// 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 );
```

References

[Reynolds87] C. Reynolds. Flocks, Herds and Schools: A Distributed Behavioral 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. Pages pp. 763-782.