MARCH 5-9
2007
SAN FRANCISCO

MOSCONE CENTER

TAKE CONTROL

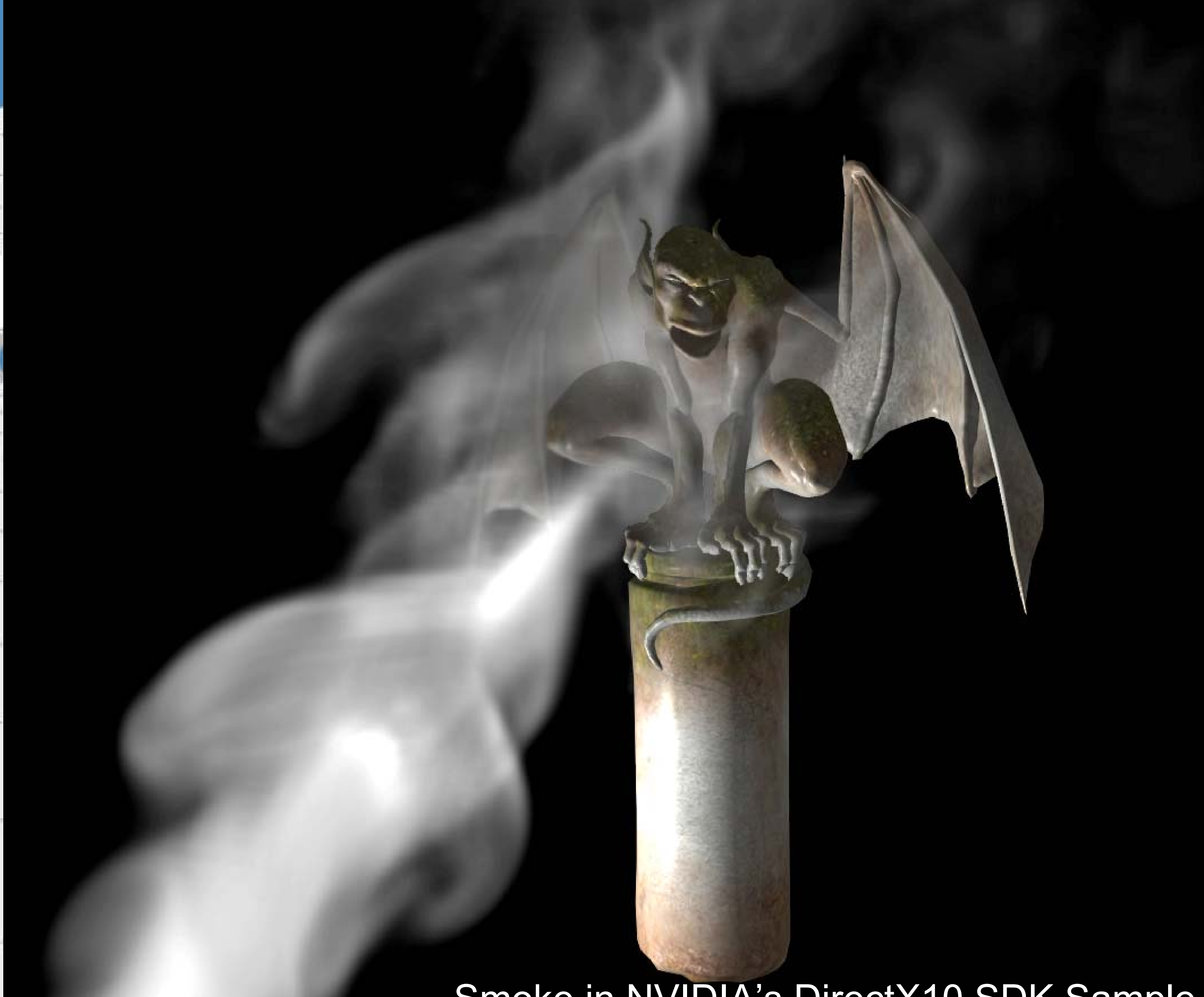
www.gdconf.com

CMP

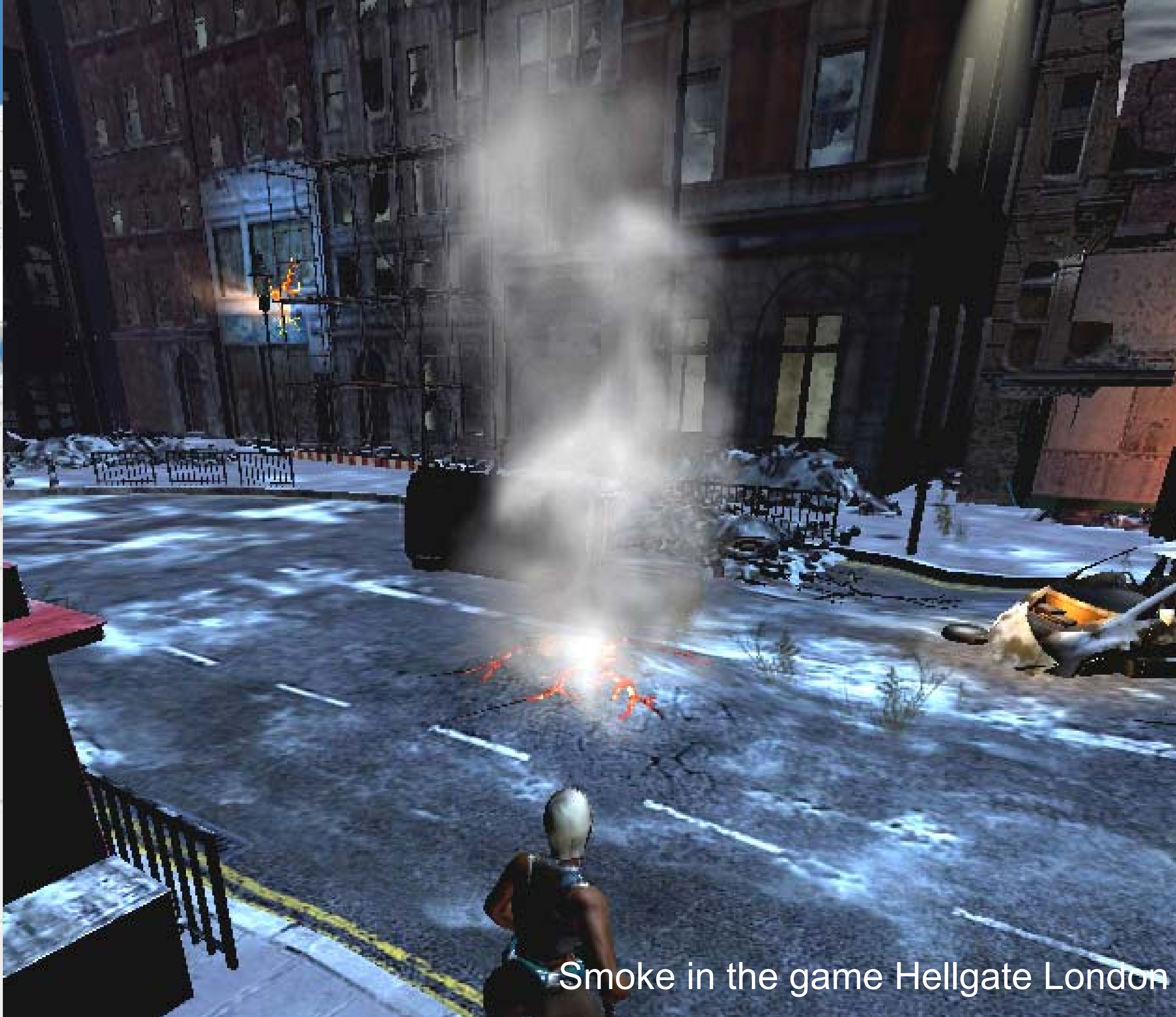
Game Developers Conference 07
TAKE CONTROL
March 5-9, 2007 in San Francisco
CMP

Real-Time Volumetric Smoke using D3D10

Sarah Tariq and Ignacio Llamas
NVIDIA Developer Technology



Smoke in NVIDIA’s DirectX10 SDK Sample



Smoke in the game Hellgate London

Talk outline:

- Why 3D fluid simulation is important
- Overview of process
- Fluid simulation basics
- Dynamic arbitrary boundaries
- Rendering
- Tradeoffs and optimizations

GameDevelopers Conference 07
TAKE CONTROL
March 5-9, 2007 in San Francisco
CMP

Why is this cool

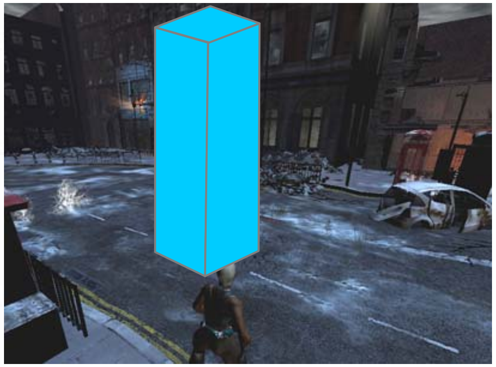
- Current approaches to rendering smoke in games have limitations
  - Video textures
  - Particle systems
- Generalizes to other fluids like water or fire

Why now?

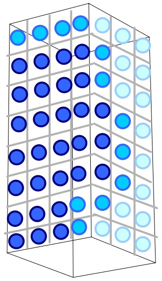
- Jos Stam 03: Real time fluids for games
  Harris03, Sander04, FluidSim on the GPU
- Sheer number of operations needed can only be supported by modern high end GPUs
- New features in DirectX10
  - Render to 3D texture
  - Geometry Shader
  - Stream Out

Overview

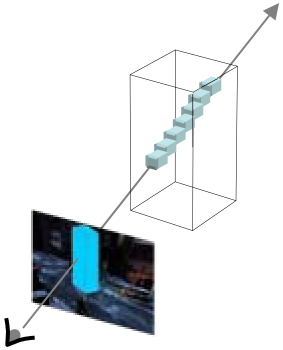
Scene



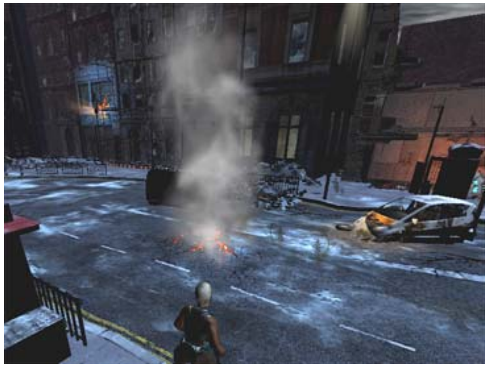
Decide where to place the smoke



Discretize space and simulate



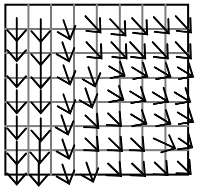
Render



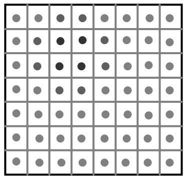
Composite on top of scene

Fluid Simulation

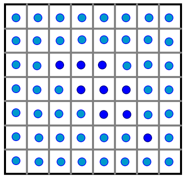
- A fluid (with constant density and temperature) is described by a velocity and pressure field
- Navier-Stokes equations mathematically defines the evolution of these fields over time; impose that the field conserves both mass and momentum
- To use these equations we discretize the space into a grid
- Define smoke density, velocity and pressure at the center of each grid cell
- At each time step, we use the equations to determine the new values of the fields



Velocity

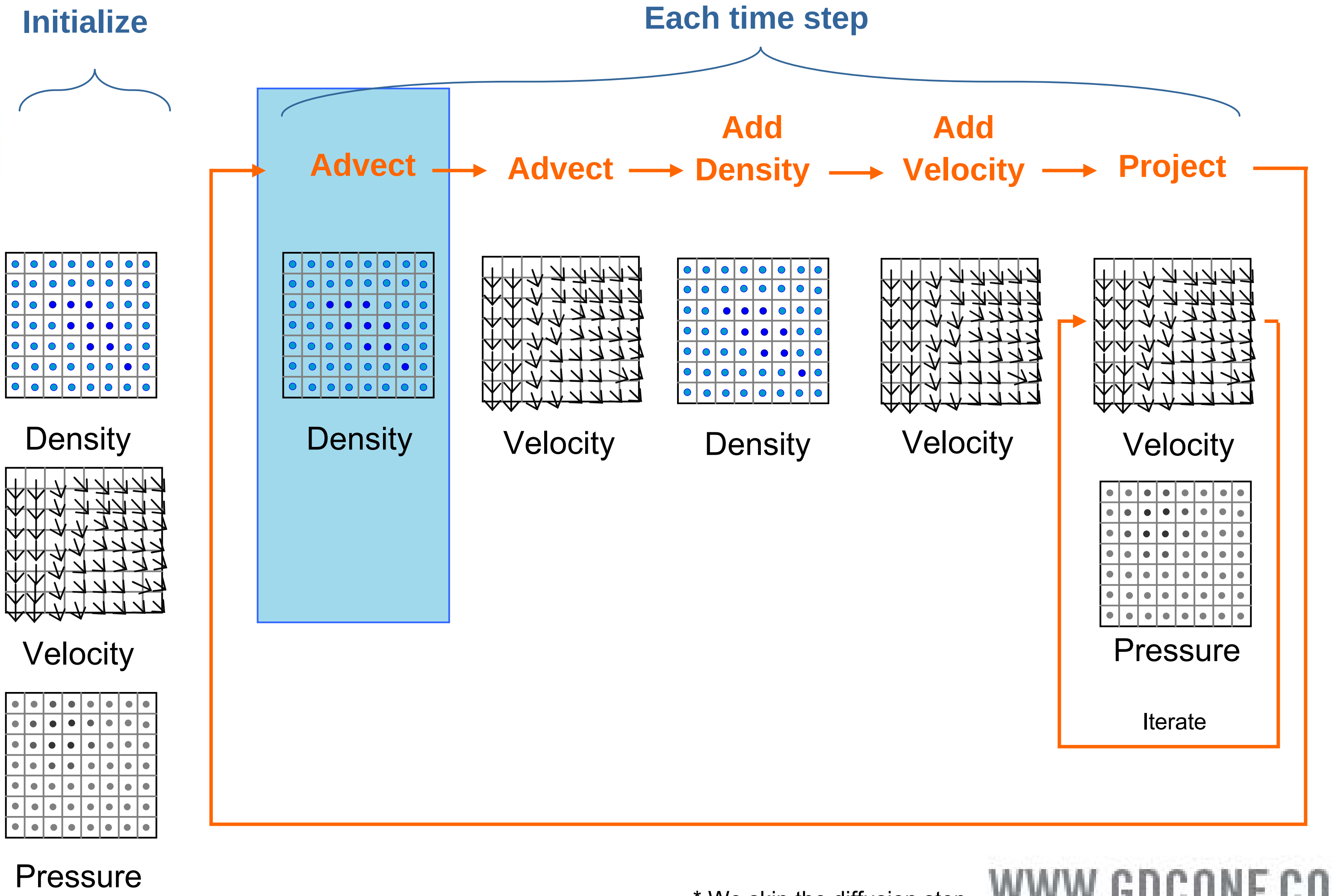


Pressure



Density

Fluid Simulation steps



Initialize

Density

Velocity

Pressure

Each time step

Advect → Advect → Add Density → Add Velocity → Project

Density

Velocity

Density

Velocity

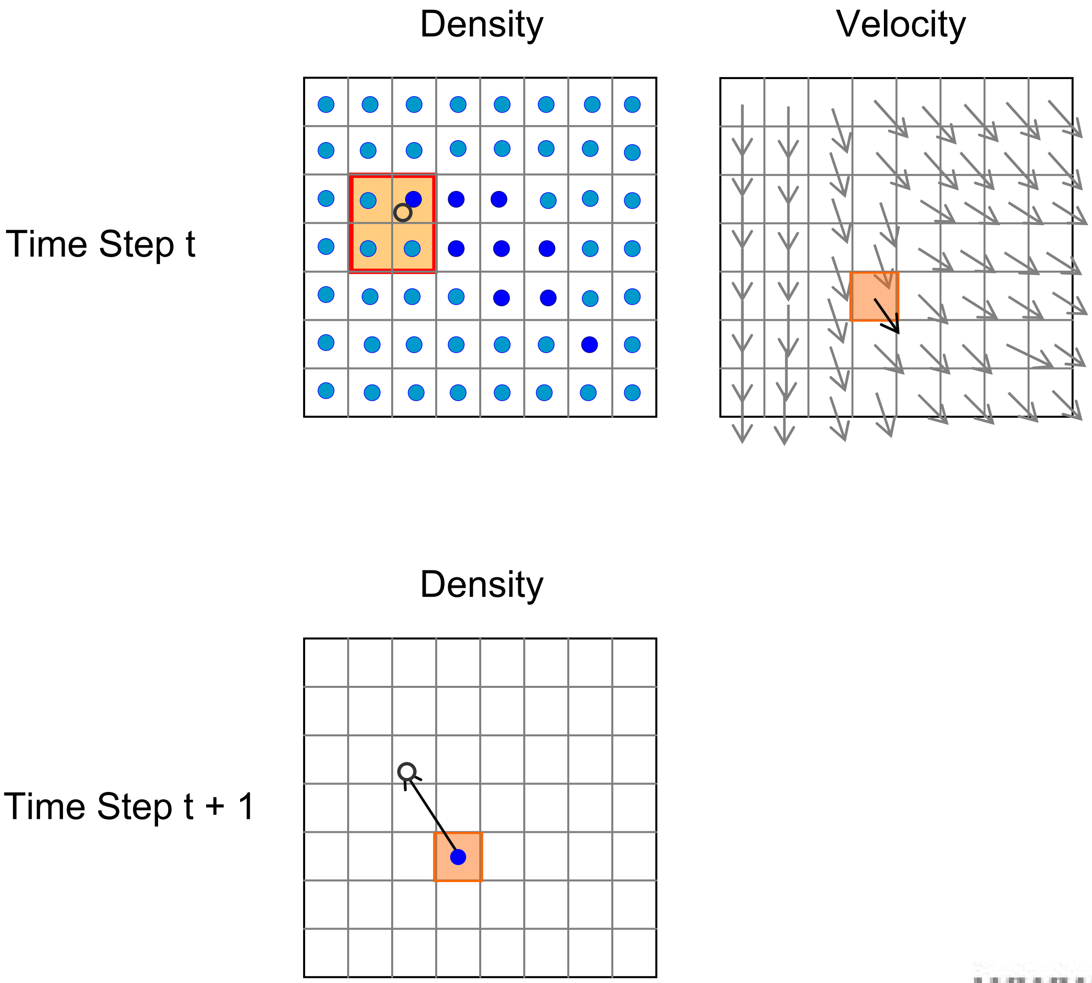
Velocity

Pressure

Iterate

* We skip the diffusion step

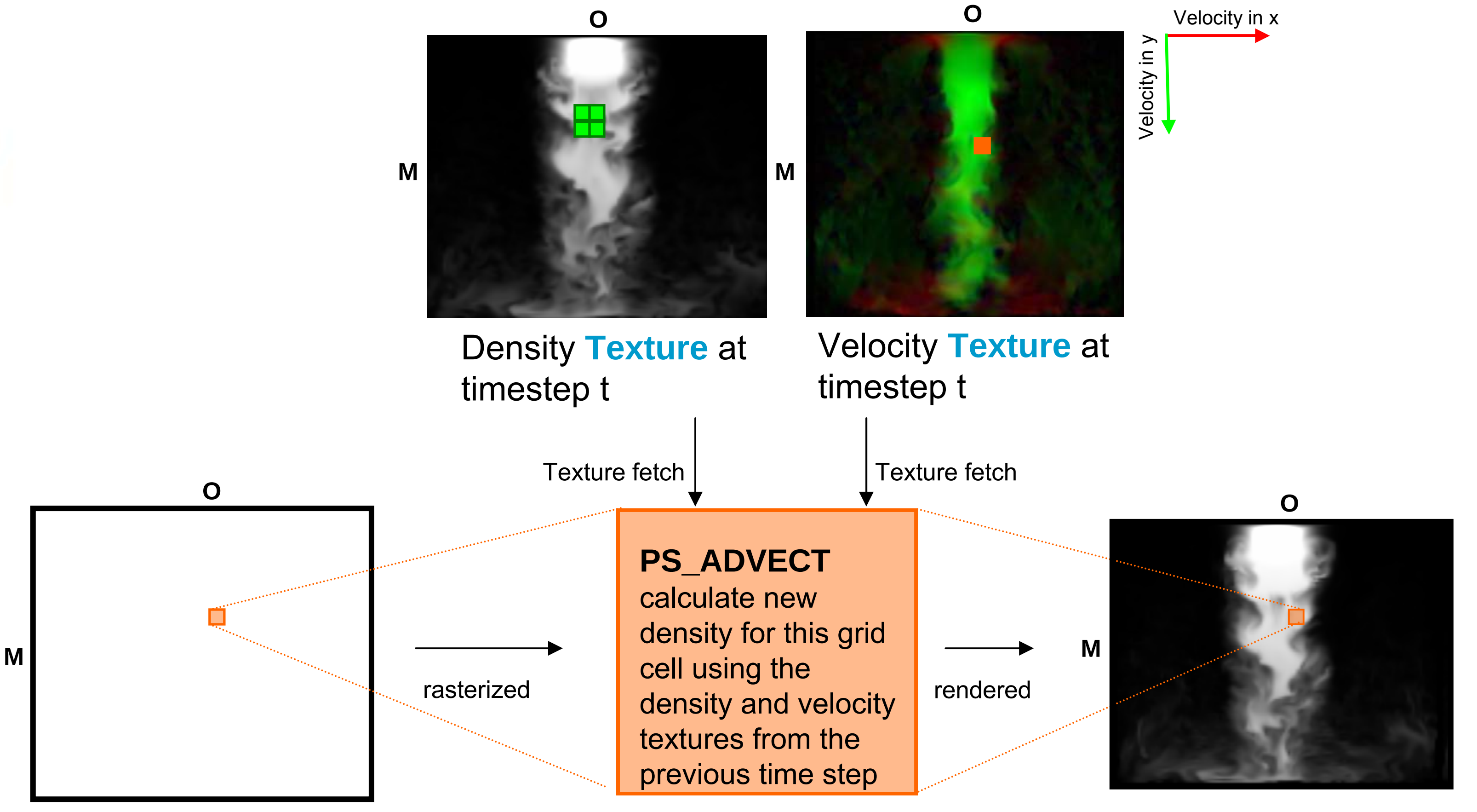
Advect



Fluid Simulation on the GPU

- Velocity, Density, Pressure → Textures
- Simulate one substep for entire grid → Render a grid sized, screen aligned, quad
- Calculations for a grid cell → Pixel Shader
- Output values → using Render to Texture

Advect on the GPU



Density **Texture** at timestep t

Velocity **Texture** at timestep t

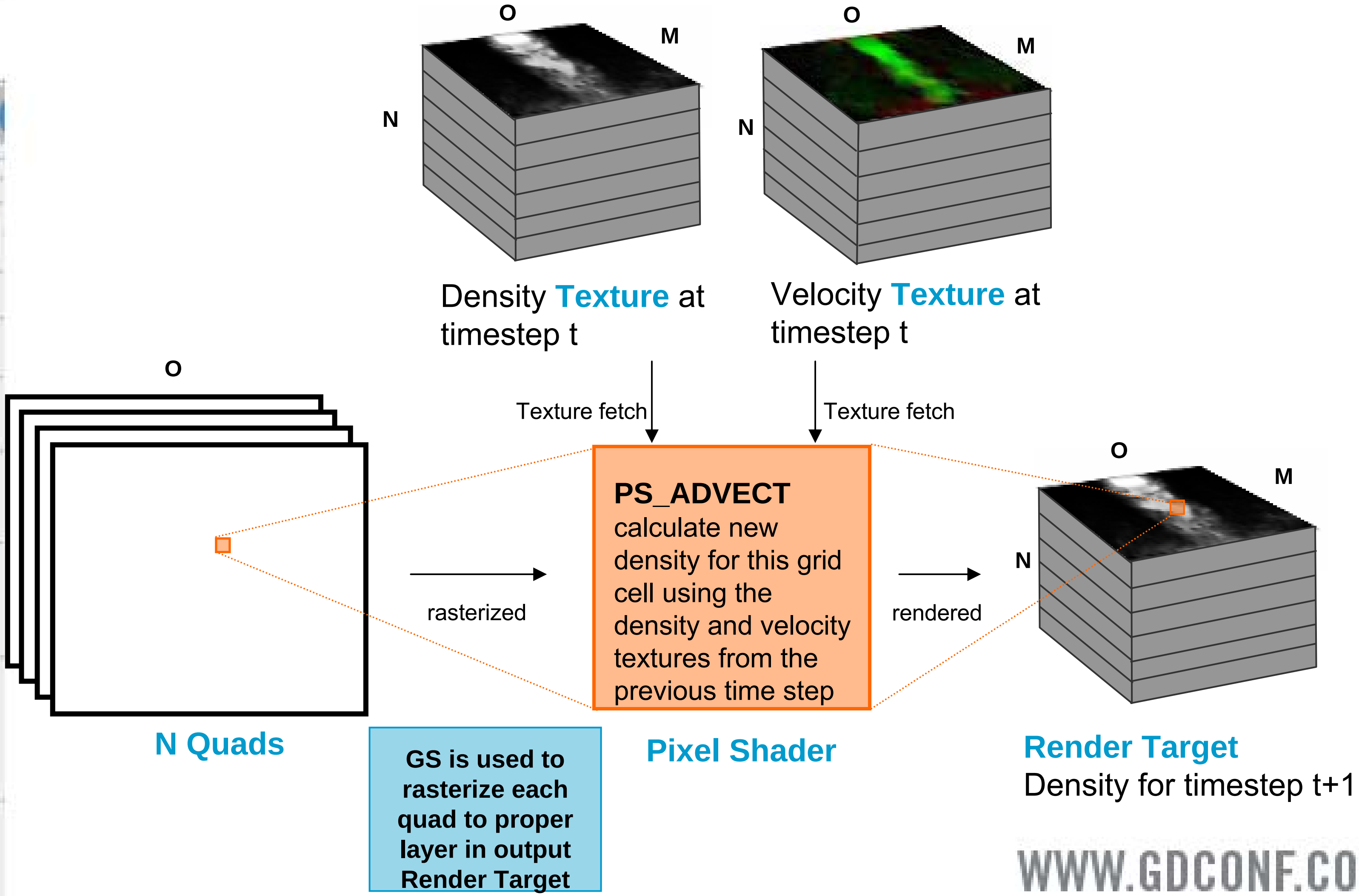
PS_ADVECT
calculate new density for this grid cell using the density and velocity textures from the previous time step

Quad

Pixel Shader

Render Target
Density for timestep t+1

Advect on the GPU in 3D



Density **Texture** at timestep t

Velocity **Texture** at timestep t

Texture fetch

Texture fetch

rasterized

PS_ADVECT
calculate new density for this grid cell using the density and velocity textures from the previous time step

rendered

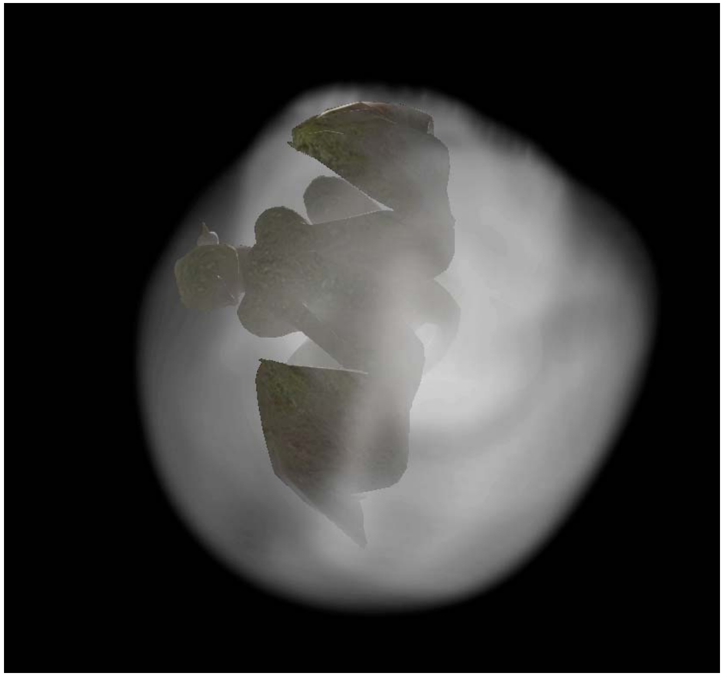
N Quads

GS is used to rasterize each quad to proper layer in output Render Target

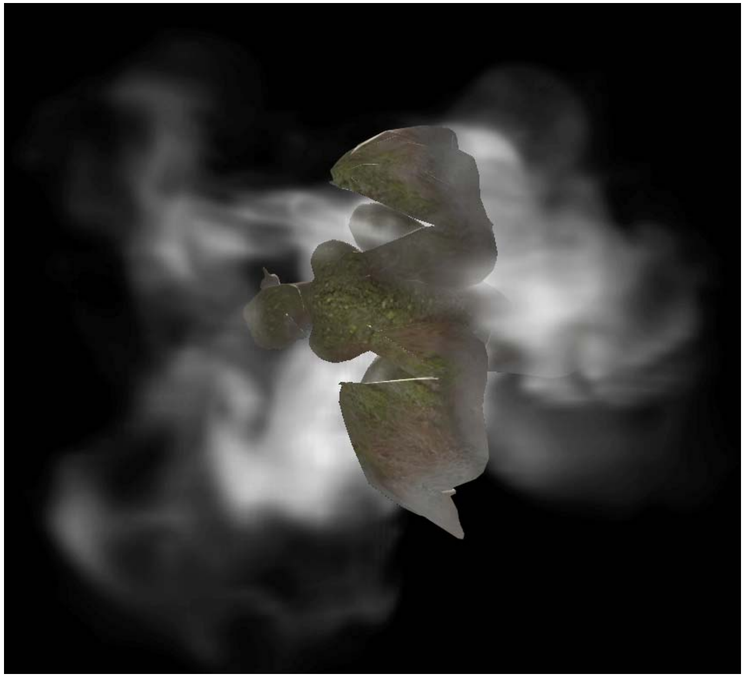
Pixel Shader

Render Target
Density for timestep t+1

Obstacles



Smoke only compositing with the scene

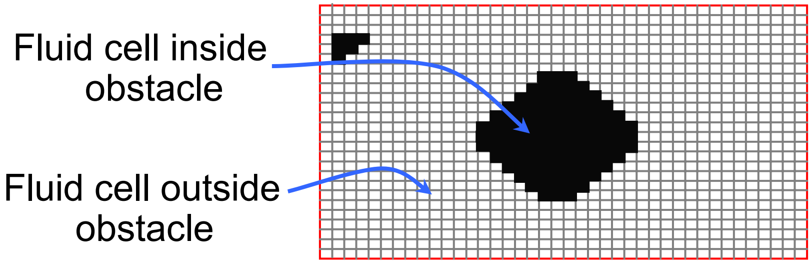


Smoke interacting with obstacles and compositing with the scene

Obstacles

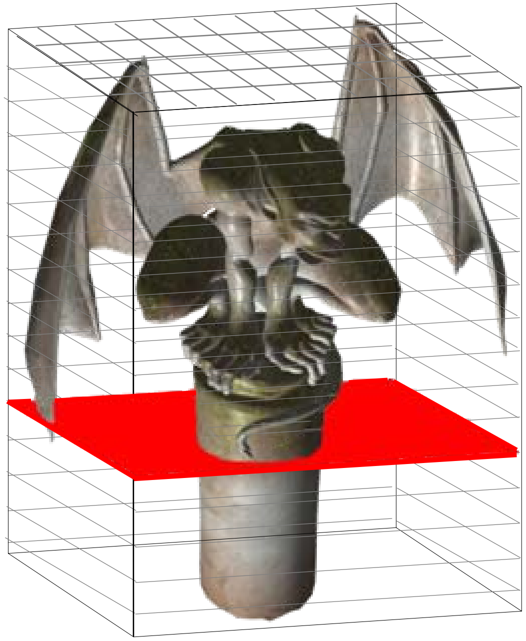
- Implicit shapes
  - Like spheres, cylinders
- Voxelize objects
  - Static : Voxelize just once, offline
  - Moving:Voxelize objects per frame

Obstacle texture



Fluid cell inside obstacle

Fluid cell outside obstacle

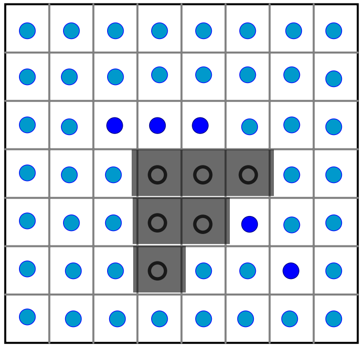


Dealing with Obstacles

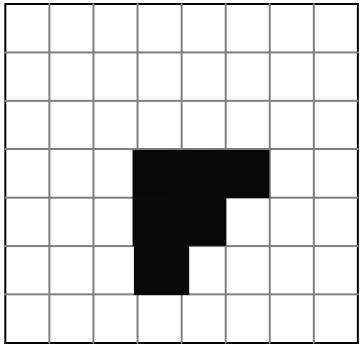
- How should the fluid react to obstacles?
  - The fluid should not enter obstacles cells
  - If the obstacles are moving they should impart the correct velocity on the fluid
- How the fluid reacts to the obstacles → *Boundary Conditions*

Boundary Conditions for Density

- No density should be added to or advected into the interior of obstacles



Density

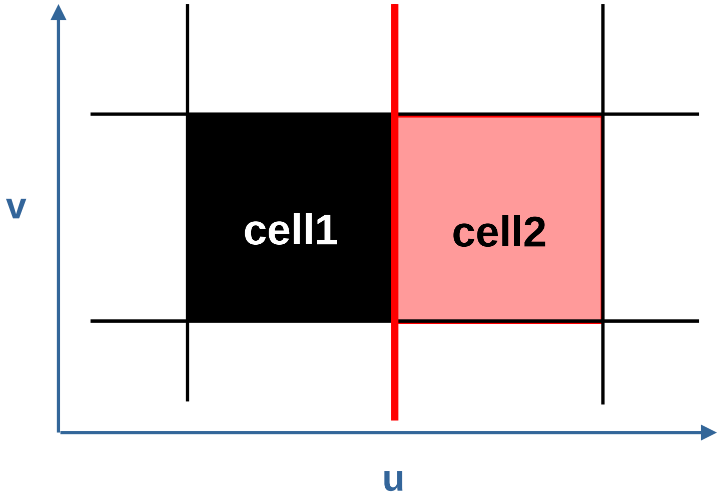


Obstacles

Boundary Conditions for Pressure

- Derivative of the pressure across the boundary should be zero

(Neumann boundary conditions - specifying derivative)



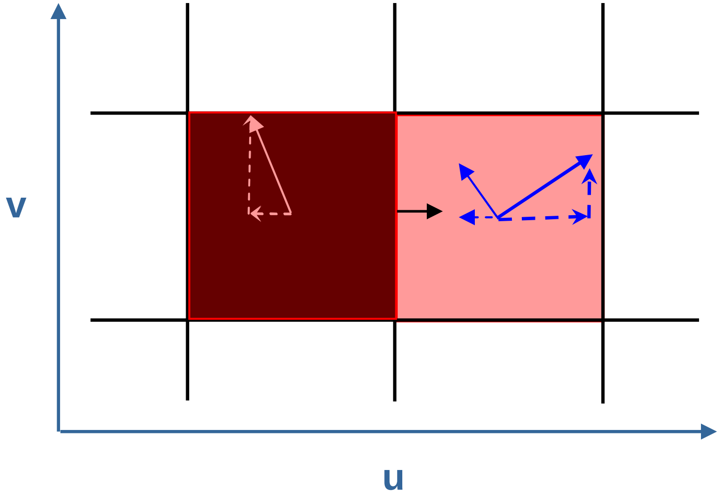
PressureCell1 – PressureCell2 = 0

PressureCell1 = PressureCell2

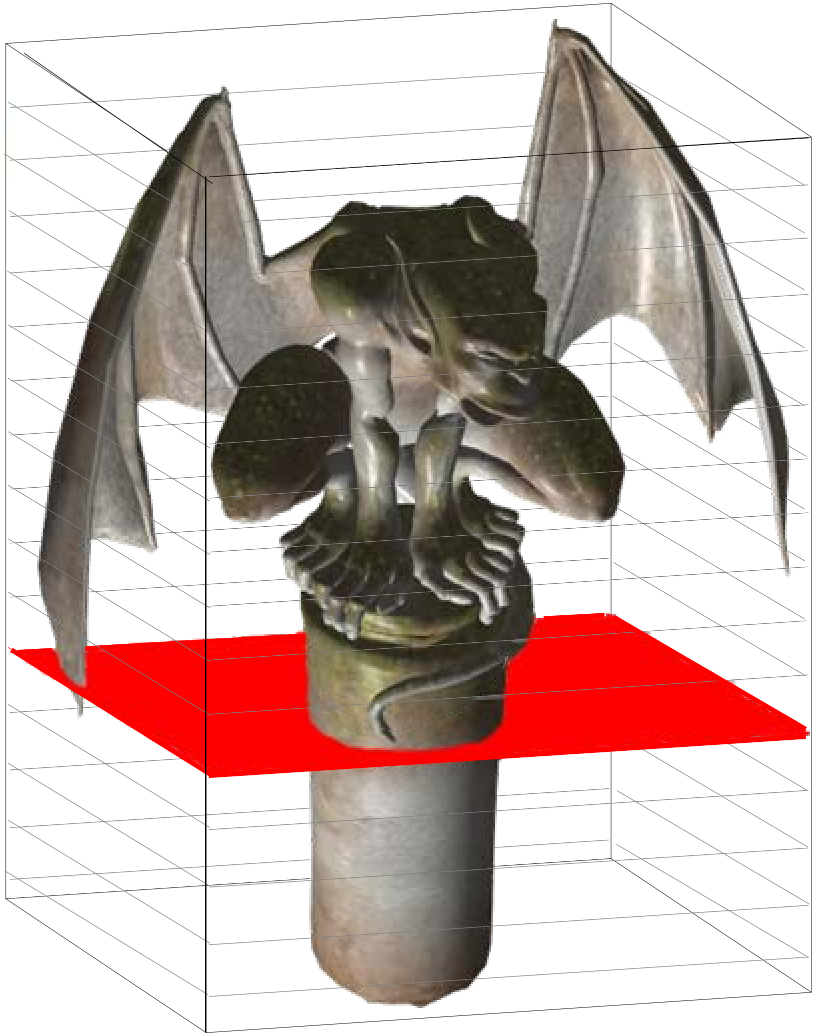
Boundary Conditions for Velocity

- The velocity normal to the boundary of the obstacle should be equal for fluid and obstacle

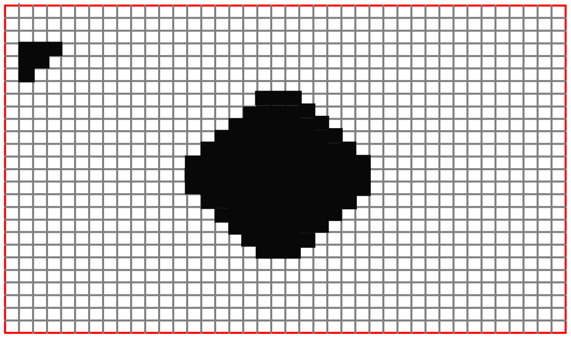
(Dirichlet boundary conditions – specifying value)



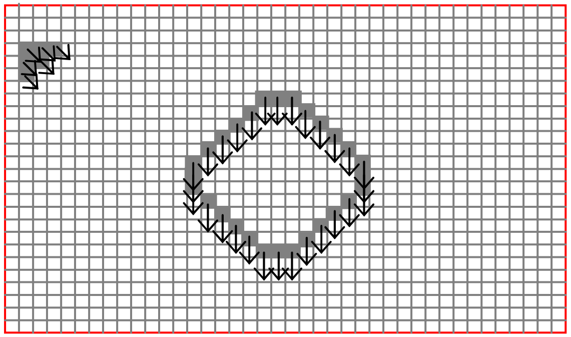
Voxelizing an object



Obstacle texture

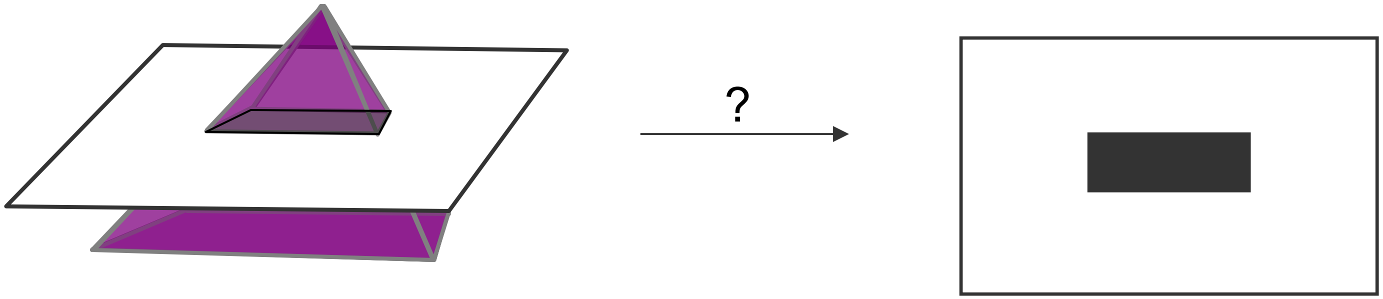


Obstacle Velocity texture

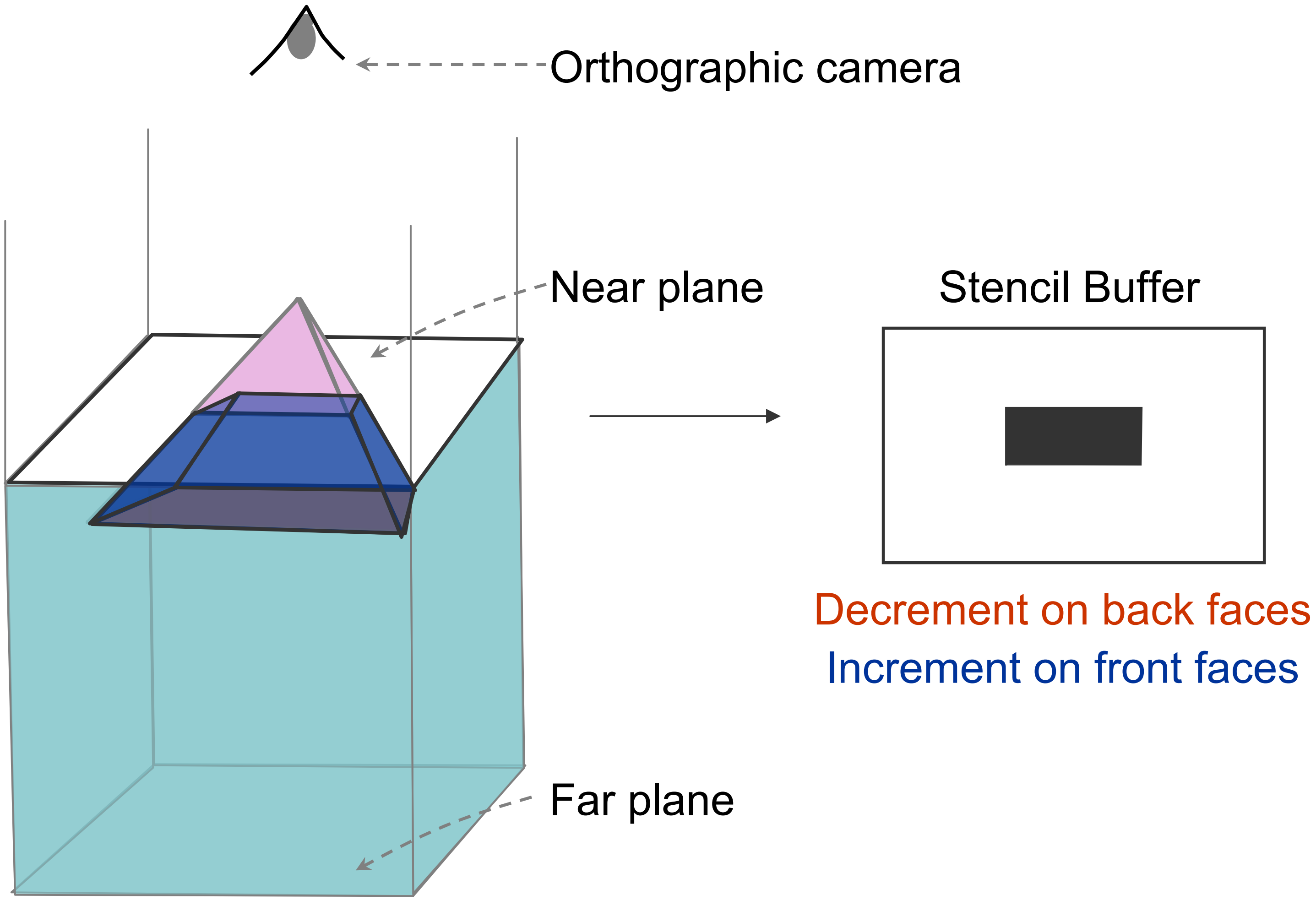


Use low res collision model for voxelization

Voxelizing a simple object



Voxelizing a simple object

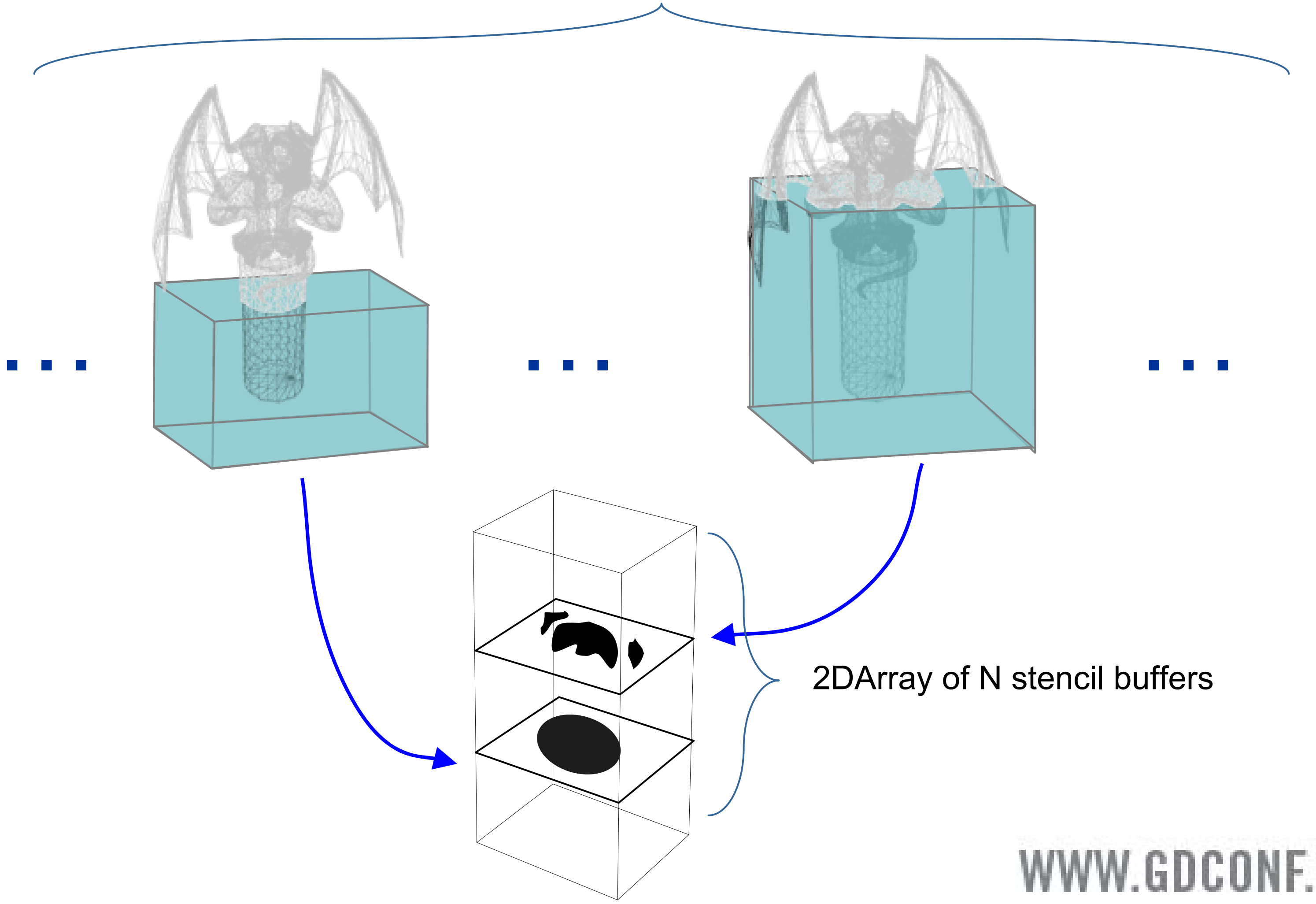


Decrement on back faces

Increment on front faces

Voxelization

Render model N times, each time with a different near plane

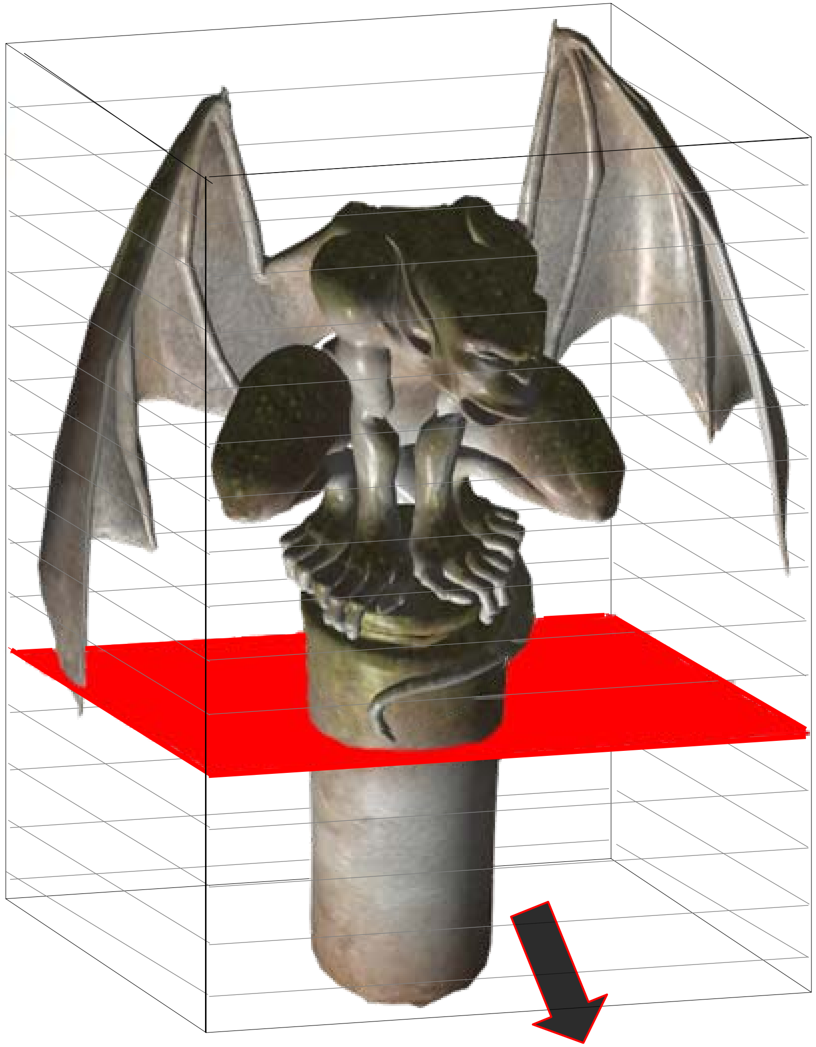


2DArray of N stencil buffers

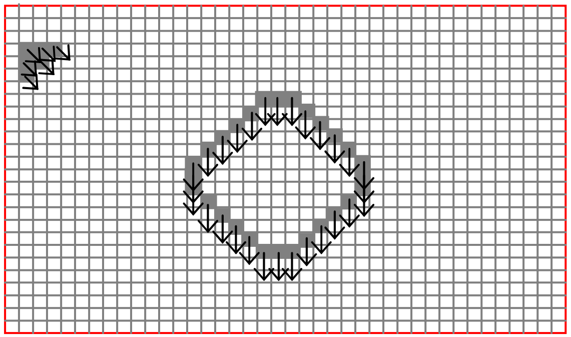
Optimizations

- Skin the mesh once per frame, *stream out* the skinned mesh, and rendered n times using instancing
- Each instance uses a different projection matrix with the appropriate near plane
- Each instance is rasterized to a different slice in the output 3D obstacle texture using the geometry shader.

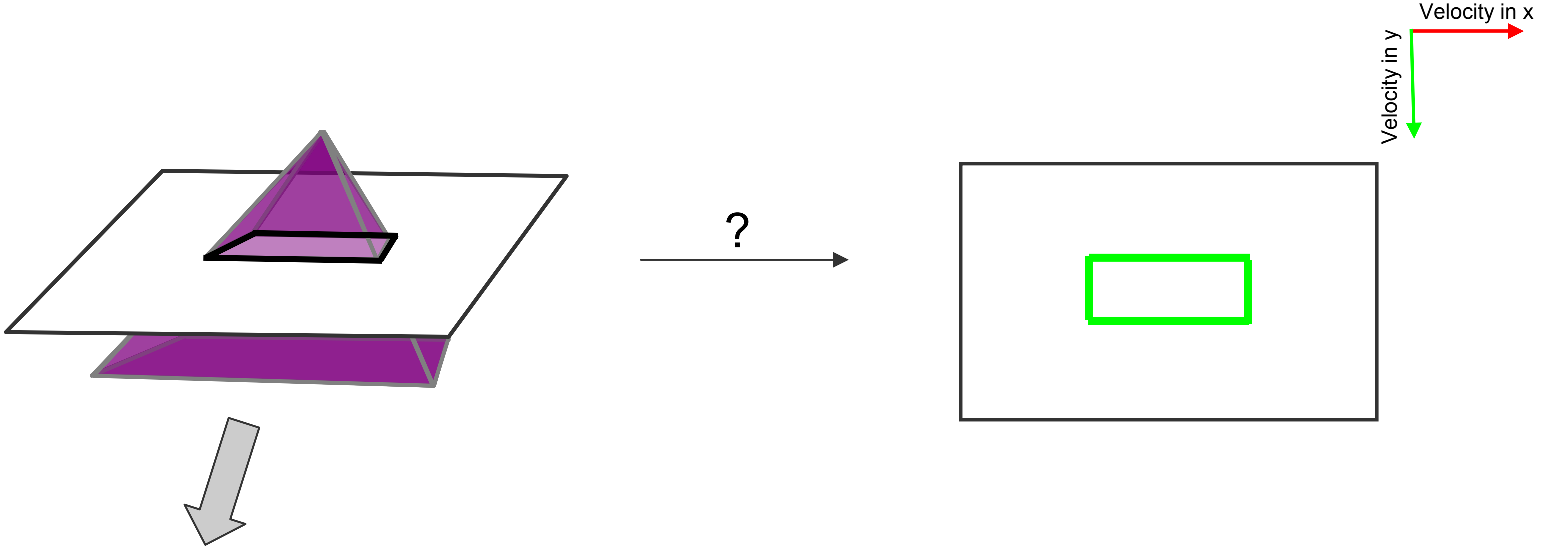
Voxelizing velocity



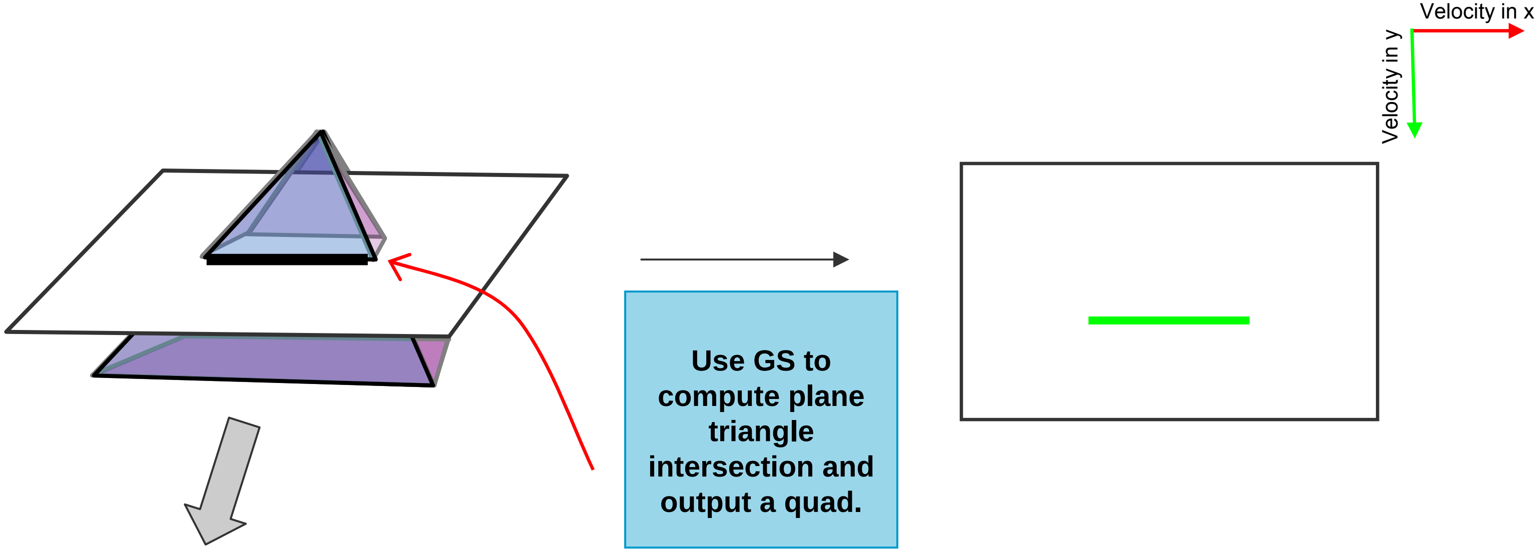
Obstacle Velocity texture



Moving obstacles



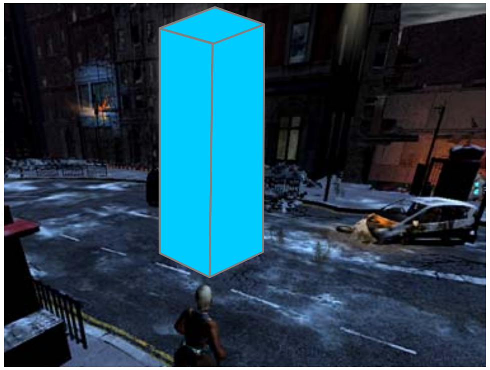
Moving obstacles



Use GS to compute plane triangle intersection and output a quad.

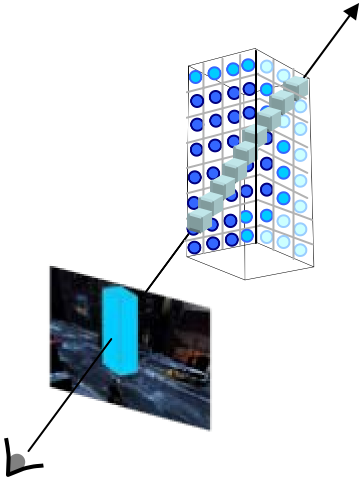
The quad is rendered with the interpolated velocities of the vertices (derived by subtracting current vertex positions from previous ones)

Rendering



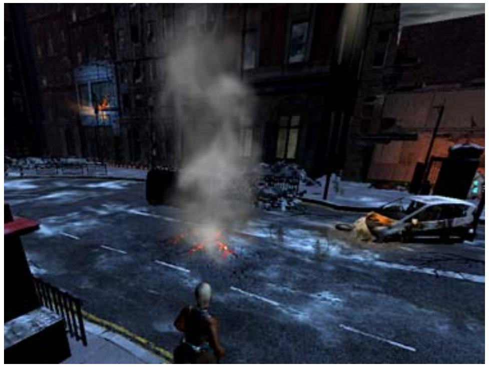
Render front faces of box

→



Raycast into the 3D Density texture

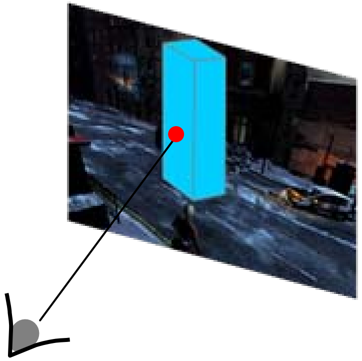
→



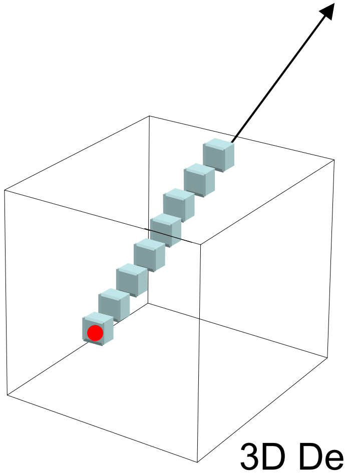
Composite into scene

Raycasting into 3D texture

What we have



What we don't have



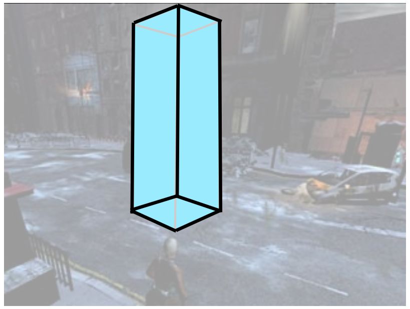
- Ray from eye to box --- Transform from world to texture space ---> - Ray in texture space

- Ray box intersection --- Transform from world to texture space ---> - Ray entry point in the texture

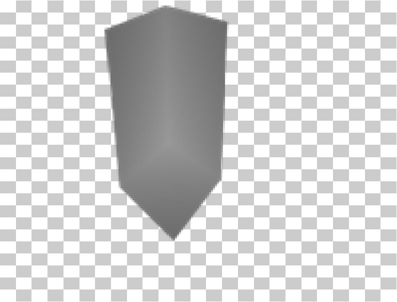
- Distance the eye ray traverses through the box --- Transform from world to grid space ---> - Number of voxels the ray traverses = Number of samples to take

Raycasting setup

Render back faces
of the box
To RayDataTexture

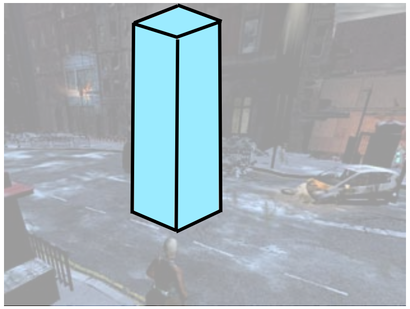


Float4(0,0,0,depthInViewSpace)

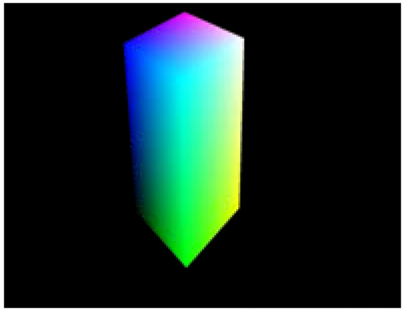


RayDataTexture.a

Render front faces
of the box
To RayDataTexture



Float4(-posInGrid, depthInViewSpace)
With subtractive (DST - SRC) blending

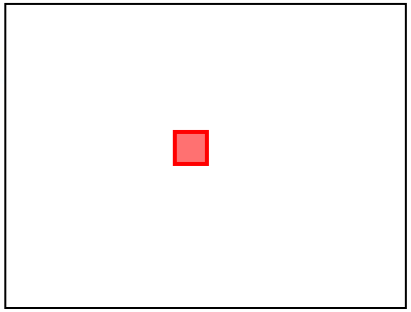


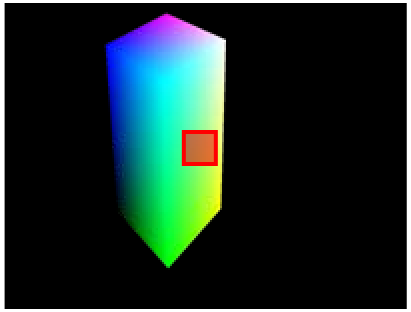
RayDataTexture.rgb

Raycasting: blending

○ = Trilinear sample from 3D texture

Render Fullscreen quad to frame buffer





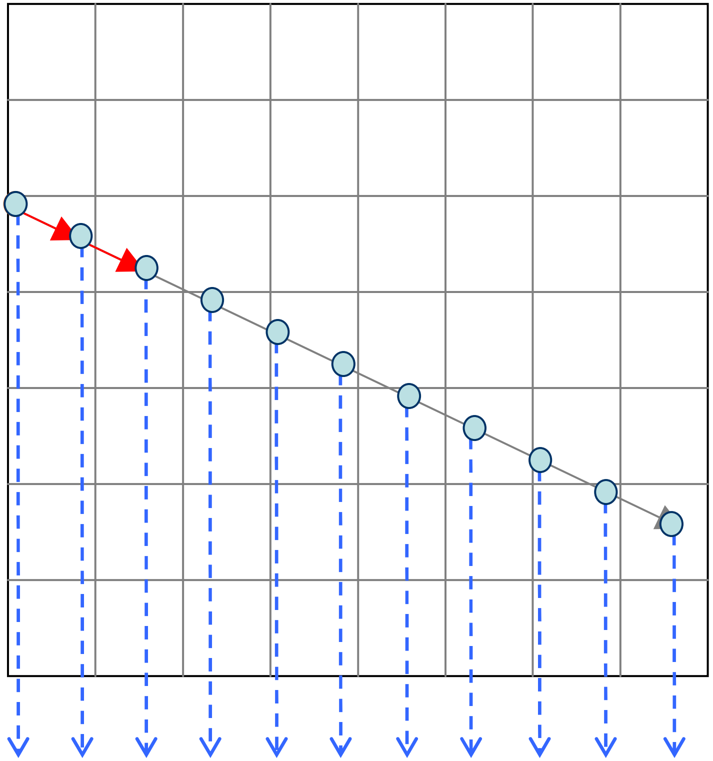
RayDataTexture

PositionInTexture = TransformToTexSpace (RayDataTexture.rgb)

MarchingVector = TransformToTexSpace (eye - RayDataTexture.rgb)

NumberOfSamples = TransformToGridSpace (RayDataTexture.a)

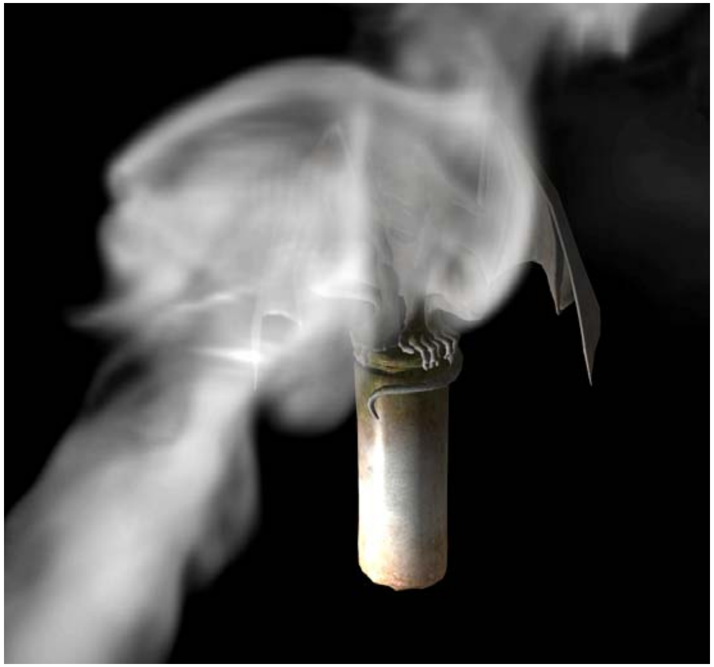
Density Texture



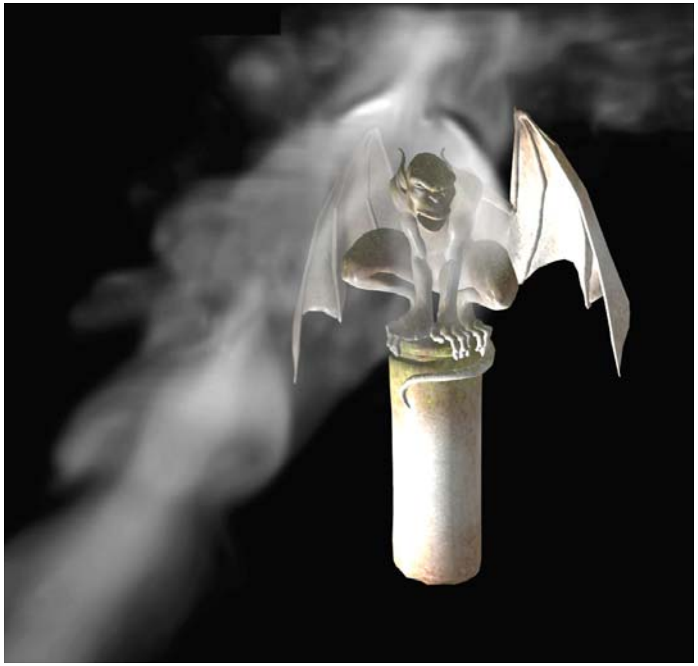
FinalColor.rgba = 0

FinalColor.rgb += sampleColor.rgb * SampleColor.a *(1.0 – FinalColor.a)
FinalColor.a += SampleColor.a * (1.0 – FinalColor.a)

Occluding the scene

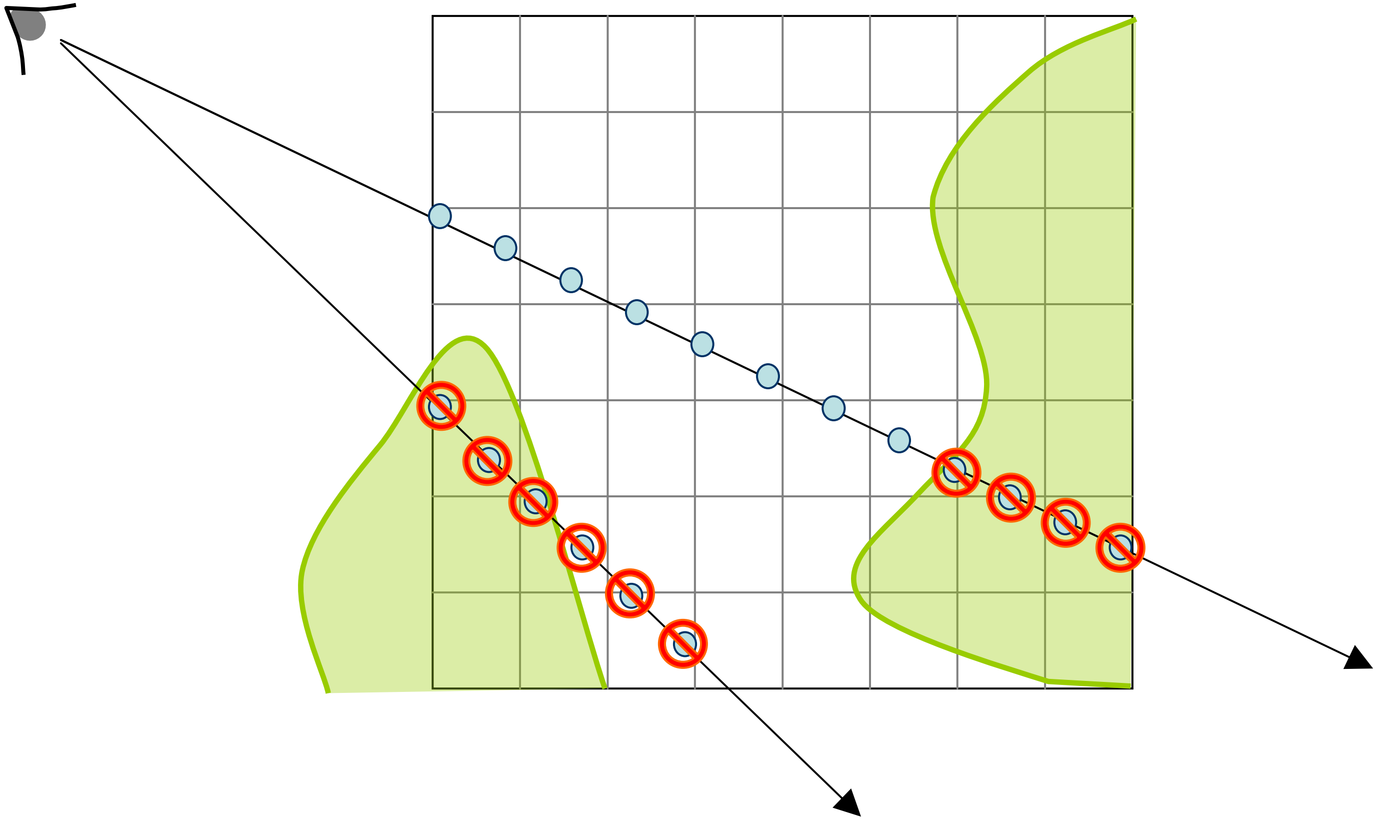


Smoke directly blended on top of the scene



Smoke correctly compositing with the scene

Integrating scene depth

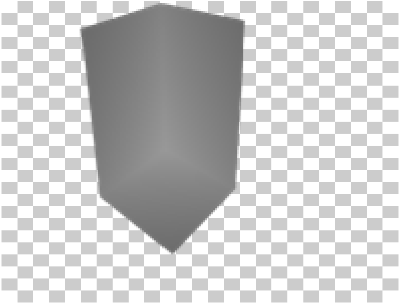


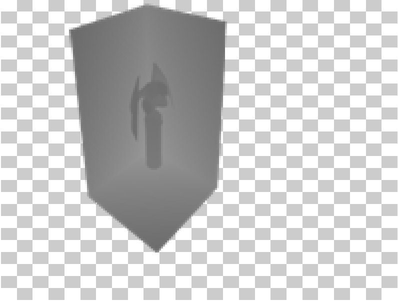
Integrating scene depth

Before

After using scene depth

Back Faces

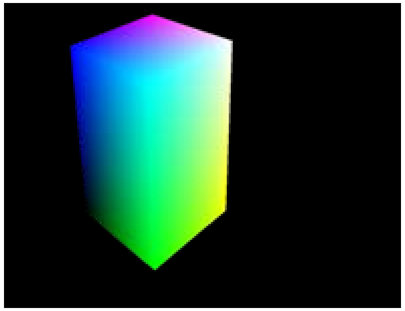


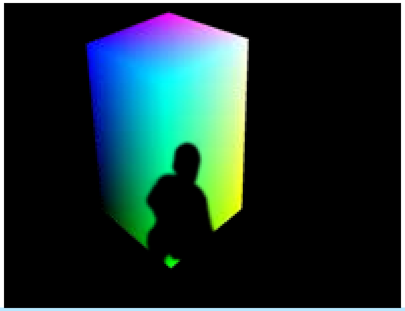


float4(0,0,0,depthInViewSpace)

float4(0,0,0,**min(sceneDepth,depthInViewSpace)**)

Front Faces

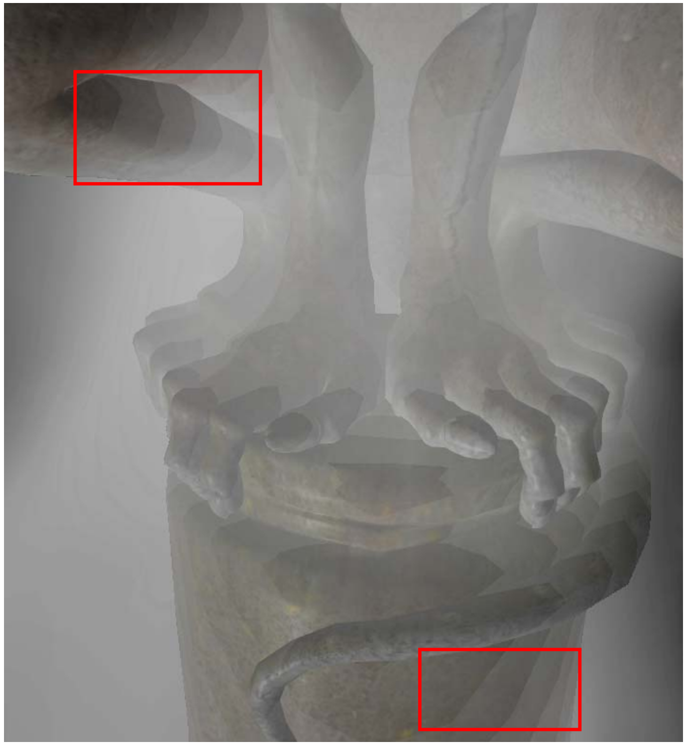




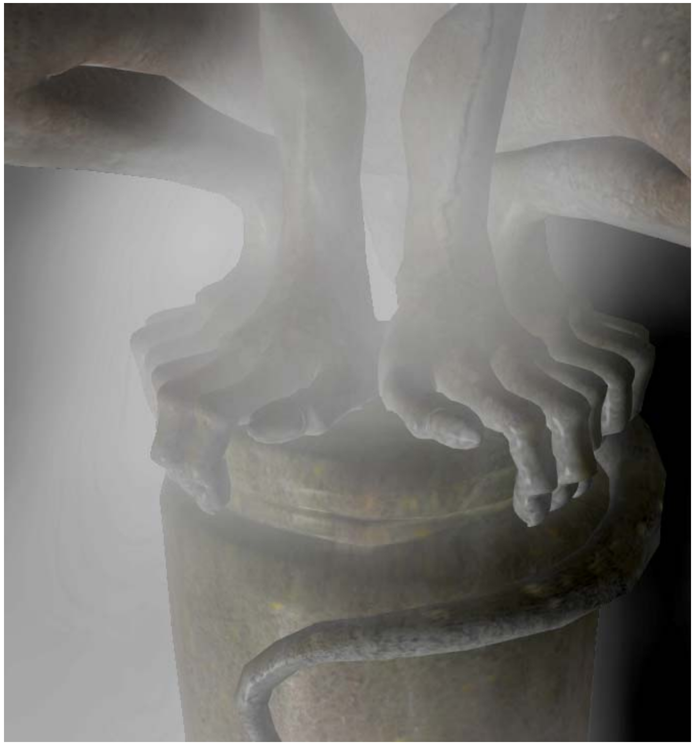
float4(-posInGrid,depthInViewSpace)

if(sceneDepth < depthInViewSpace)
float4(0,0,0,0)
float4(-posInGrid,depthInViewSpace)

Artifacts

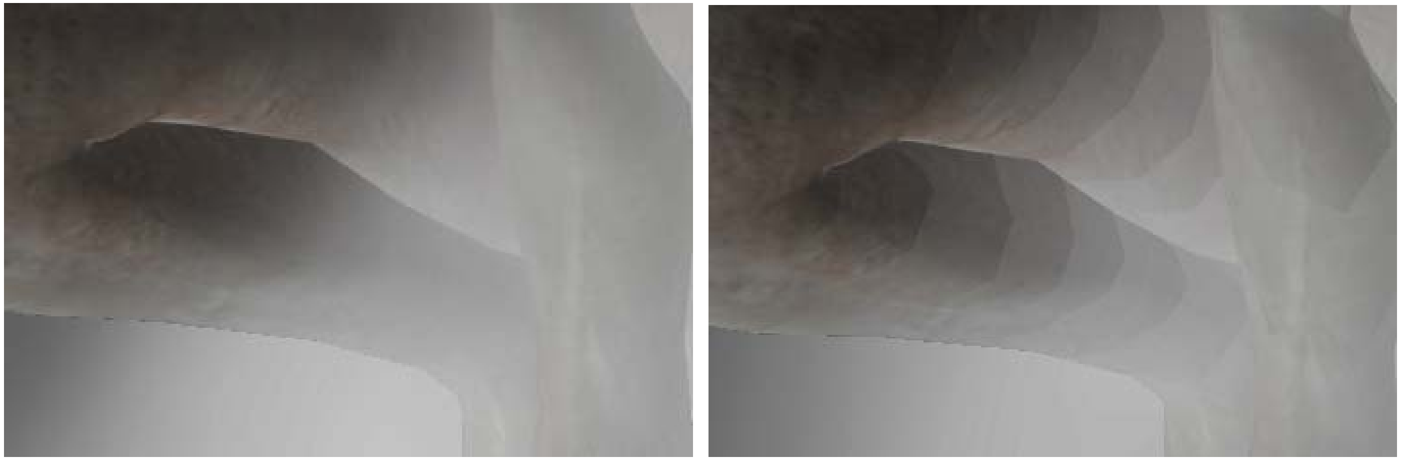


Artifacts resulting from an integral number of samples

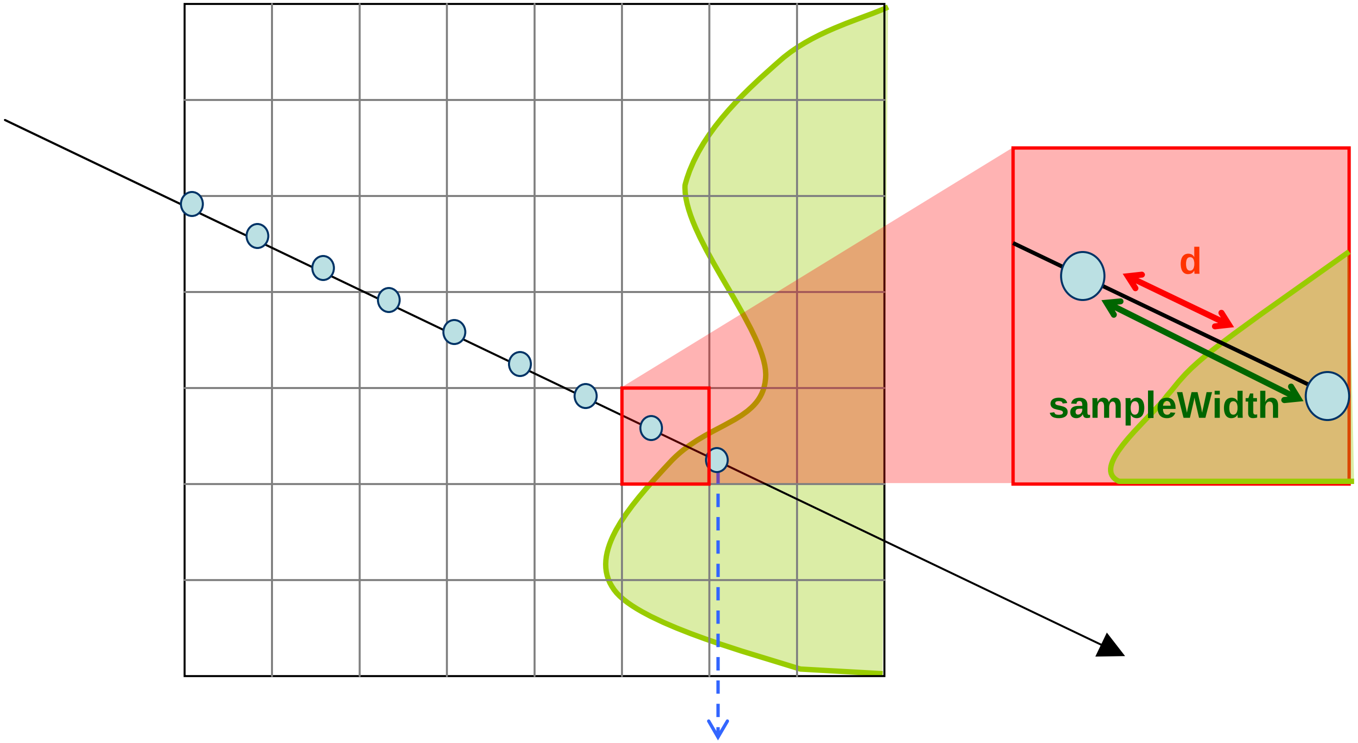


Correctly using the depth by weighted sampling

Artifacts

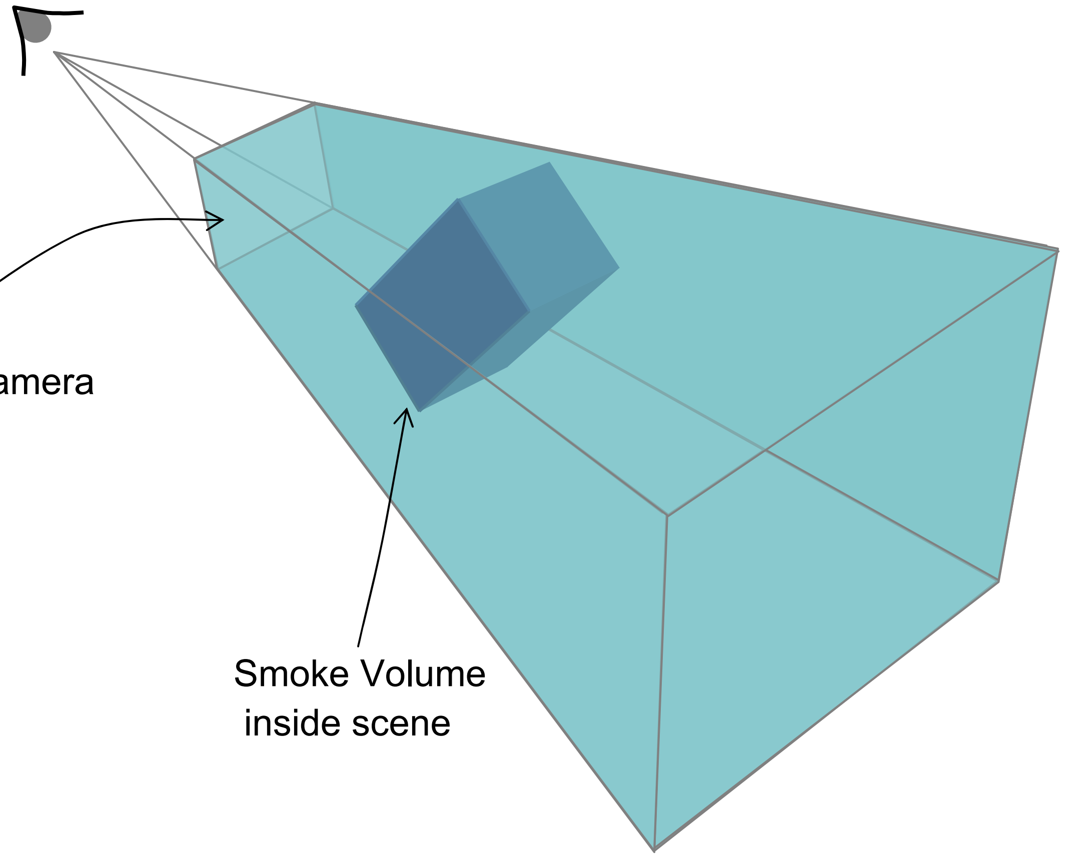


Correctly integrating scene depth by weighting the last sample



FinalColor.rgb += d/sampleWidth * SampleColor.rgb * SampleColor.a * (1.0 – FinalColor.a)
FinalColor.a += d/sampleWidth * SampleColor.a * (1.0 – FinalColor.a)

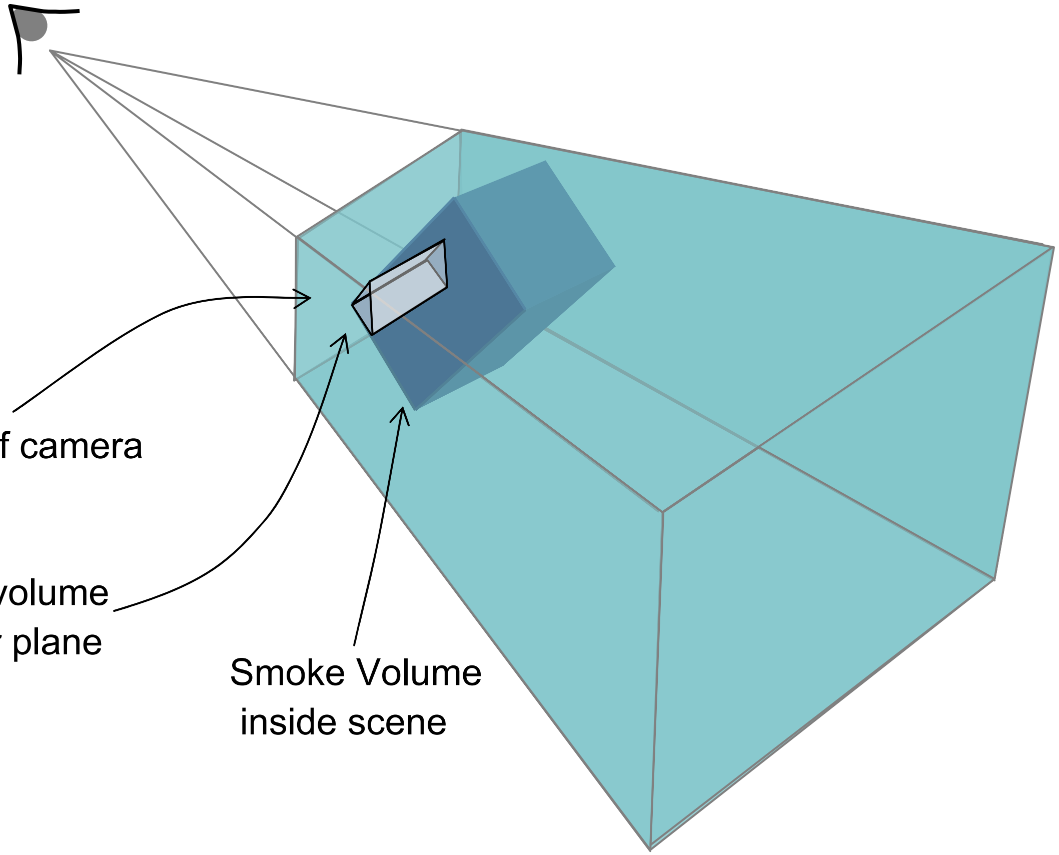
Camera inside smoke volume



Near plane of camera

Smoke Volume
inside scene

Camera inside smoke volume



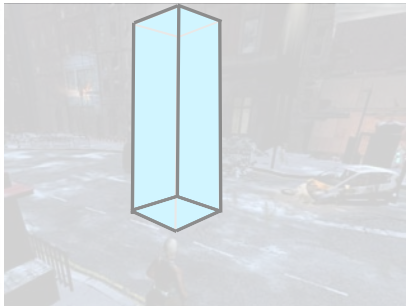
Near plane of camera

Part of smoke volume clipped by near plane

Smoke Volume inside scene

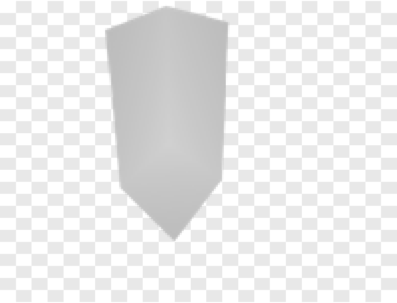
Camera inside smoke volume

Render back faces
of the box
To RayDataTexture



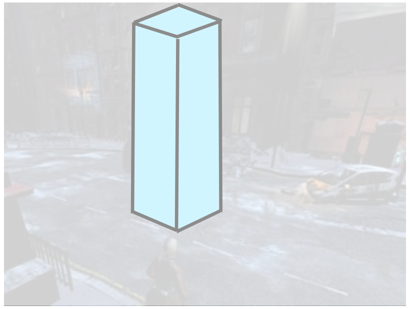
Float4(0,0,0,depthInViewSpace)

↓



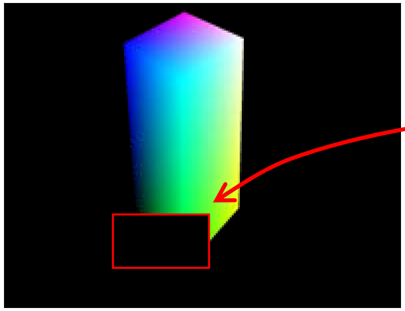
RayDataTexture.a

Render front faces
of the box
To RayDataTexture



Float4(-posInGrid, depthInViewSpace)
With subtractive (DST - SRC) blending

↓



RayDataTexture.rgb

Front faces clipped by near plane;

Depth at these pixels is incorrect
No information about the pixel's position in grid

Camera inside smoke volume

- Mark pixels where back faces have been rendered but not front
- In the raycasting step, for these marked pixels we explicitly set the position in the grid, and also subtract ZNear from the depth

Space Requirements

	Total Space	Exclusive	Shared
Simulation	32 bytes per cell	12 bytes per cell 1 x RGBA16 2 x R16	20 bytes per cell 2 x RGBA16 2 x R16
Voxelization	9 bytes per cell	-	9 bytes per cell 1 x RGBA16 1 x R8
Rendering	20 bytes per pixel	-	20 bytes per pixel 1 x RGBA32 1 x R32

Space Requirements for demo

Grid Size: 70 x 70 x 100
Screen Resolution : 1280 x 1024

	Total Space	Exclusive	Shared
Simulation	14.95 MB	5.6 MB	9.3 MB
Voxelization	4.2 MB	-	4.2 MB
Rendering	25 MB	-	25 MB

Optimizations

- Tradeoffs:
  - Reduce grid size
  - Reduce number of Jacobi iterations
- Early Z cull technique introduced by Sander et al, 2004
- LOD approach for simulation
- Render final smoke to a fixed sized off-screen buffer

Conclusion

- Interactive 3D fluid simulation at reasonable grid resolutions is feasible for games
- We presented here a brief overview of the entire process
- More information
  - NVIDIA DirectX10 SDK code sample
  - Upcoming GPU Gems3 article

References and acknowledgements

- NVIDIA Developer Technology team, Keenan Crane, and the developers of *Hellgate:London*

- Real-Time Fluid Dynamics for Games. Jos Stam, Alias | Wavefront. GDC 2003
- Simulation of Cloud Dynamics on Graphics Hardware. Mark Harris, W. Baxter, T. Scheurmann, A. Lastra. Eurographics Workshop on Graphics Hardware. 2003
- Fast Fluid Dynamics Simulation on the GPU. Mark Harris, NVIDIA. GPU Gems 2004
- Explicit Early-Z Culling for Efficient Fluid Flow Simulation and Rendering", Pedro V. Sander, Natalya Tatarchuk, Jason L. Mitchell, ATI Research Technical Report, August 2004
- Hardware Accelerated Voxelization. S. Fang and H. Cheng. Computers and Graphics, 2000