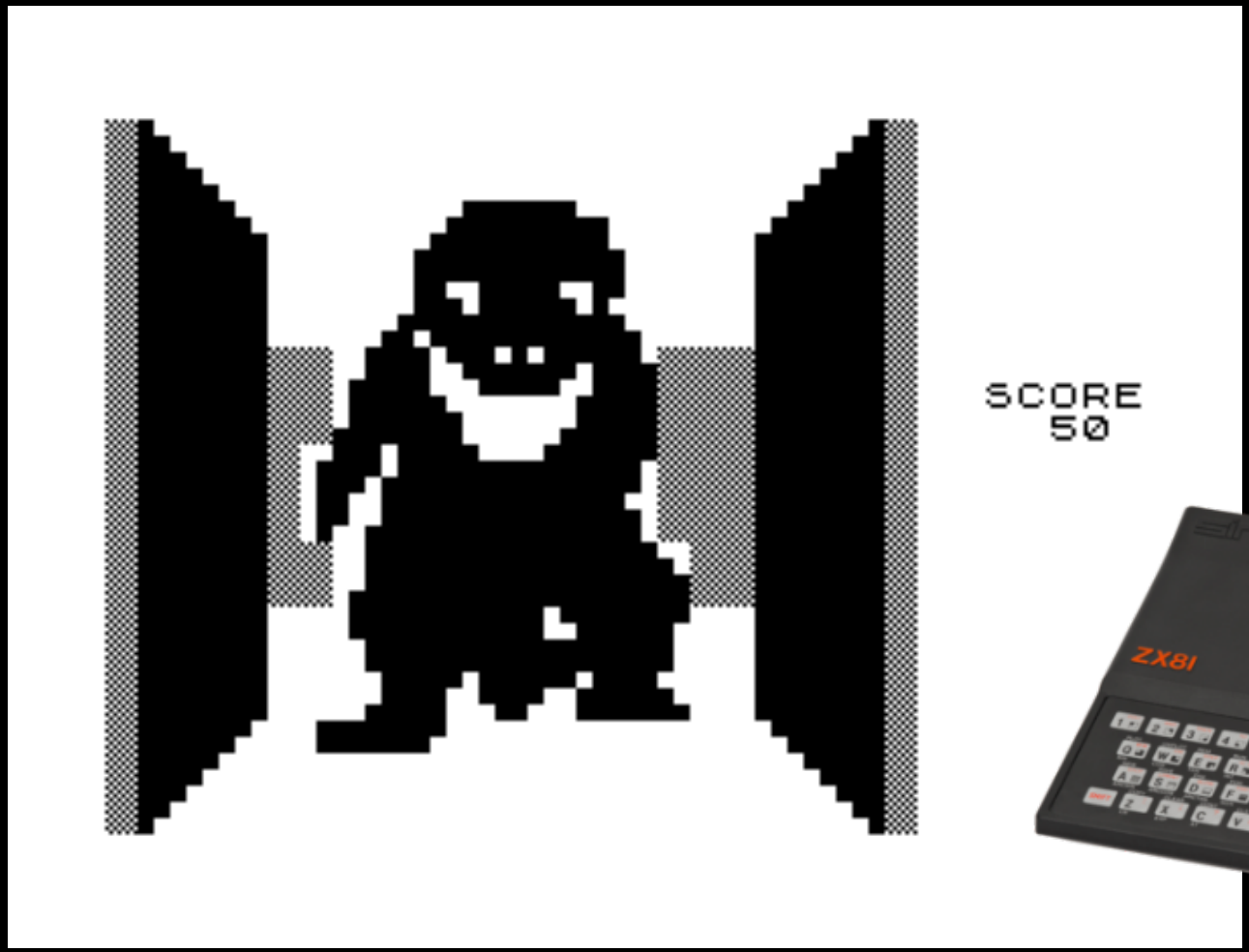
BEST OF GTC

FLAMEWORKS

REAL-TIME VOLUMETRIC FIRE & SMOKE SIMULATION

Simon Green, Principal Software Engineer

GAME GRAPHICS, 1981



BEST OF GTC

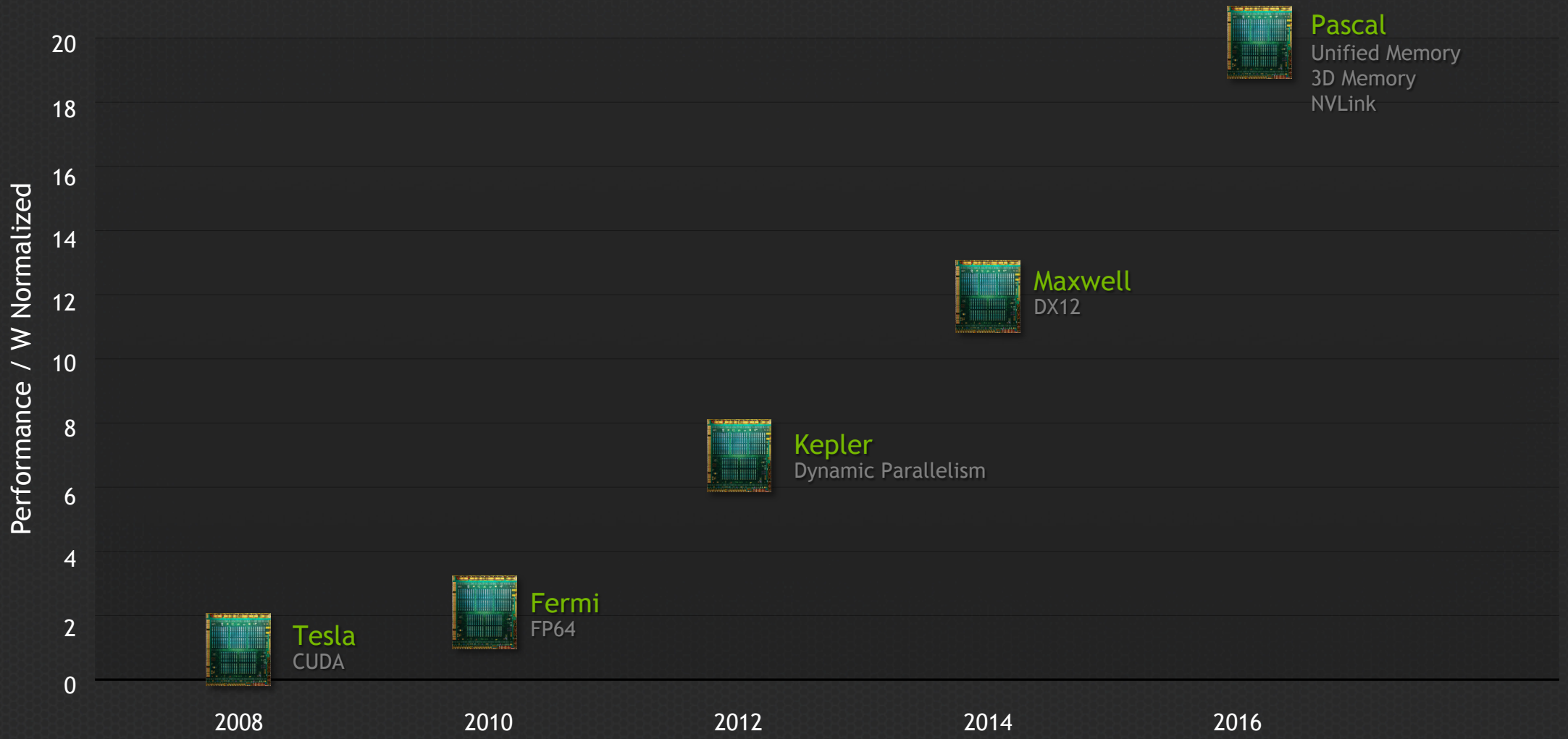
NVIDIA

BEST OF GTC

NVIDIA

GAME GRAPHICS, 2014

GPU Performance



NVIDIA GAMEWORKS™

DEVELOPER TOOLS

IDE-integrated and standalone
Debuggers, profilers and utilities

CORE SDK

Foundation for core NVIDIA technologies

VISUALFX SDK

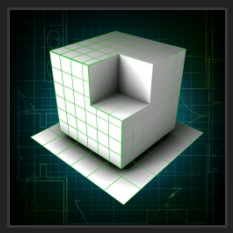
Turnkey solutions for complex, realistic effects

GRAPHICS & COMPUTE SAMPLES

Samples, documentation, tutorials organized by effect

PHYSX SDK

Most popular physics engine:
500+ games



OPTIX SDK

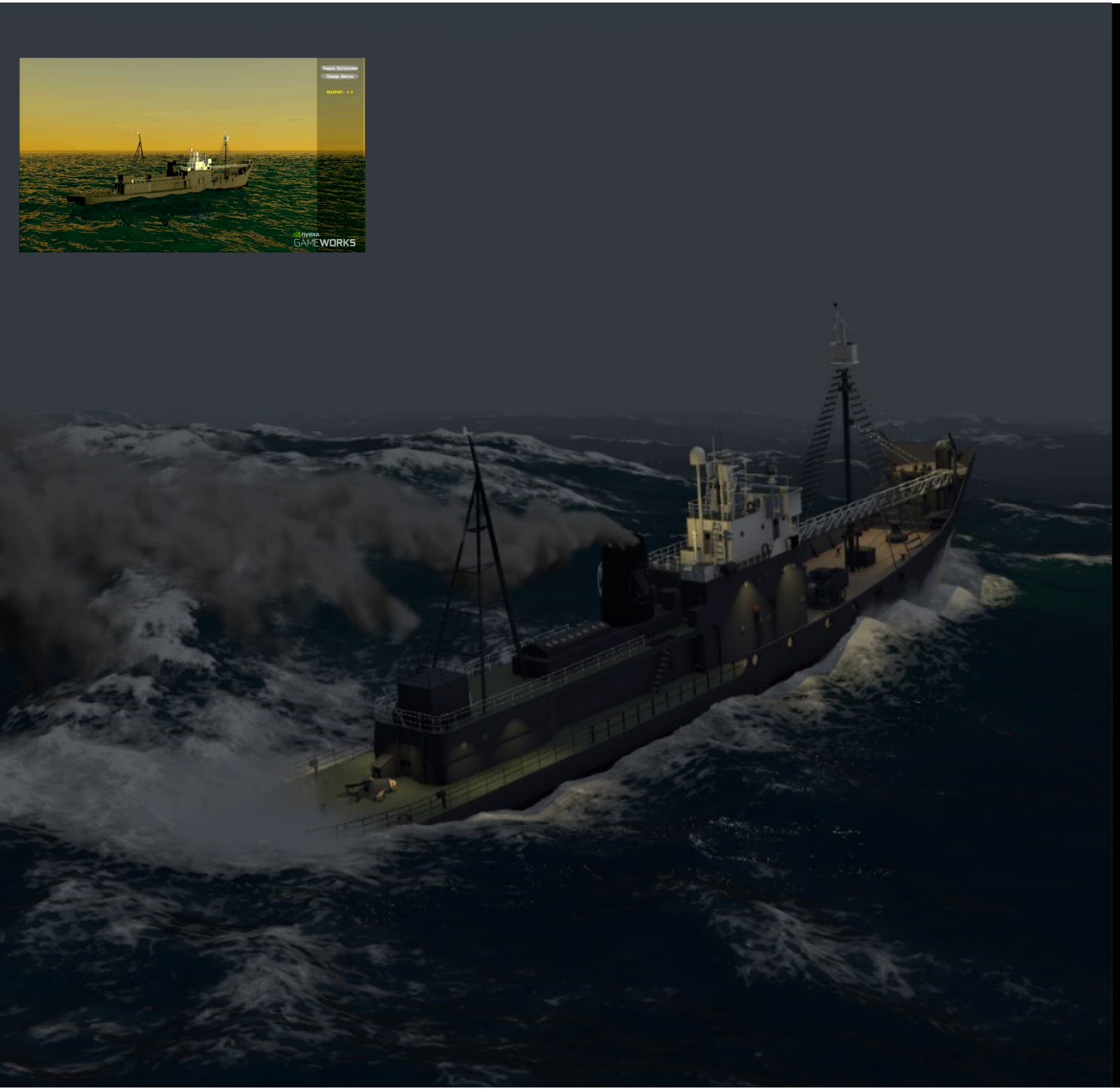
Ray tracing engine and framework

VisualFX SDK

Cinematic visual effects

Robust and easy to integrate

Multi-platform support



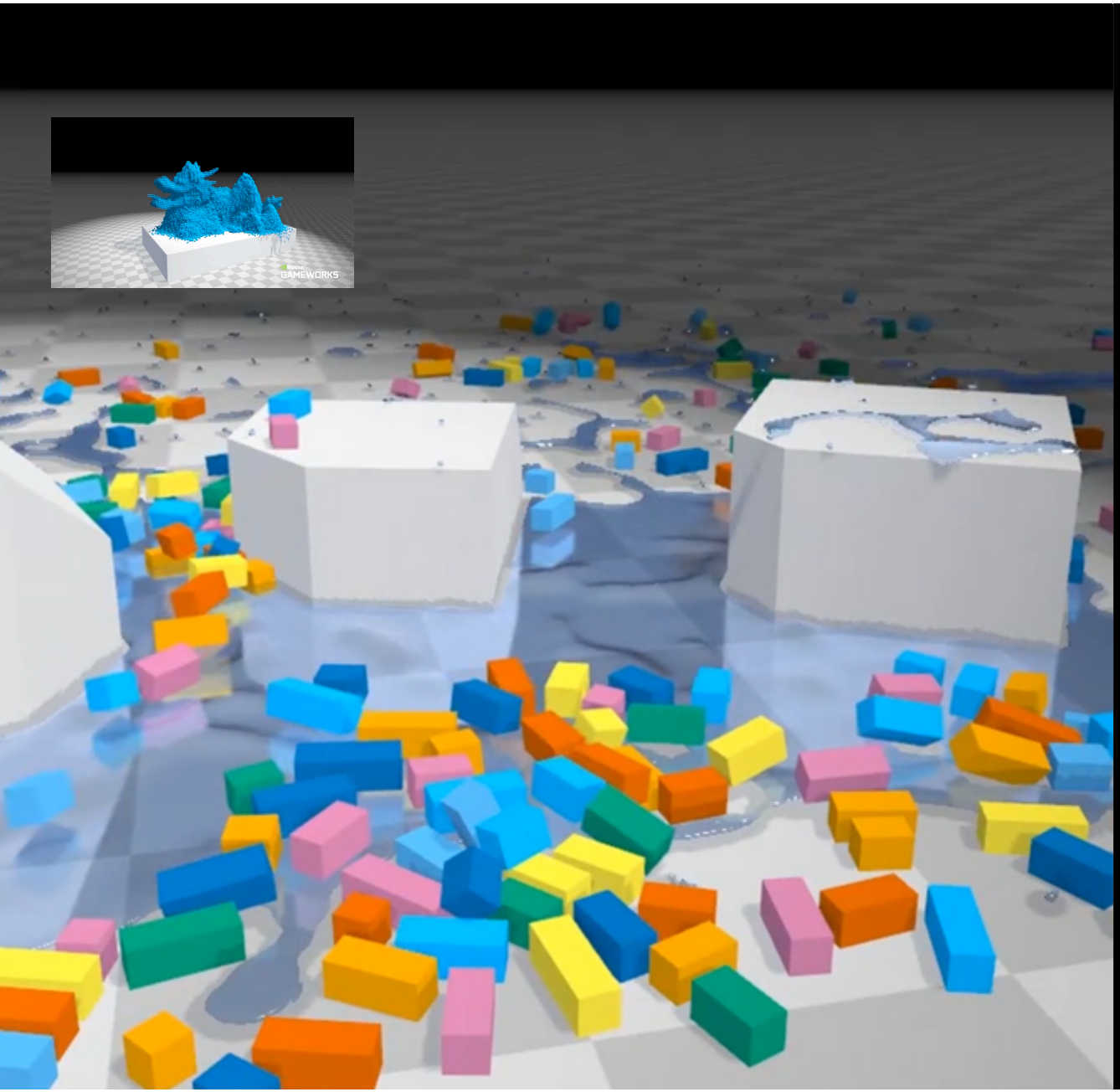
NVIDIA WaveWorks

Realistic Waves

Standalone tool

Tessendorf's spectral algorithm, based on Phillips spectrum
Multi-res simulation
Quad-tree tile-based LoDing
Host read-back
DX11 tessellation
Foam simulation
A "no graphics" path for clients (MMO servers)

PC, Steam OS, Linux, MacOS, PS4, XBOX1, Android



PhysX FleX

Unified GPU Simulation Pipeline

Unified solver for effects
Rigid/deformable bodies
Phase transition
Particles
Fluids
Cloth
Rope

Win, Linux, XBOX1/PS4, Android

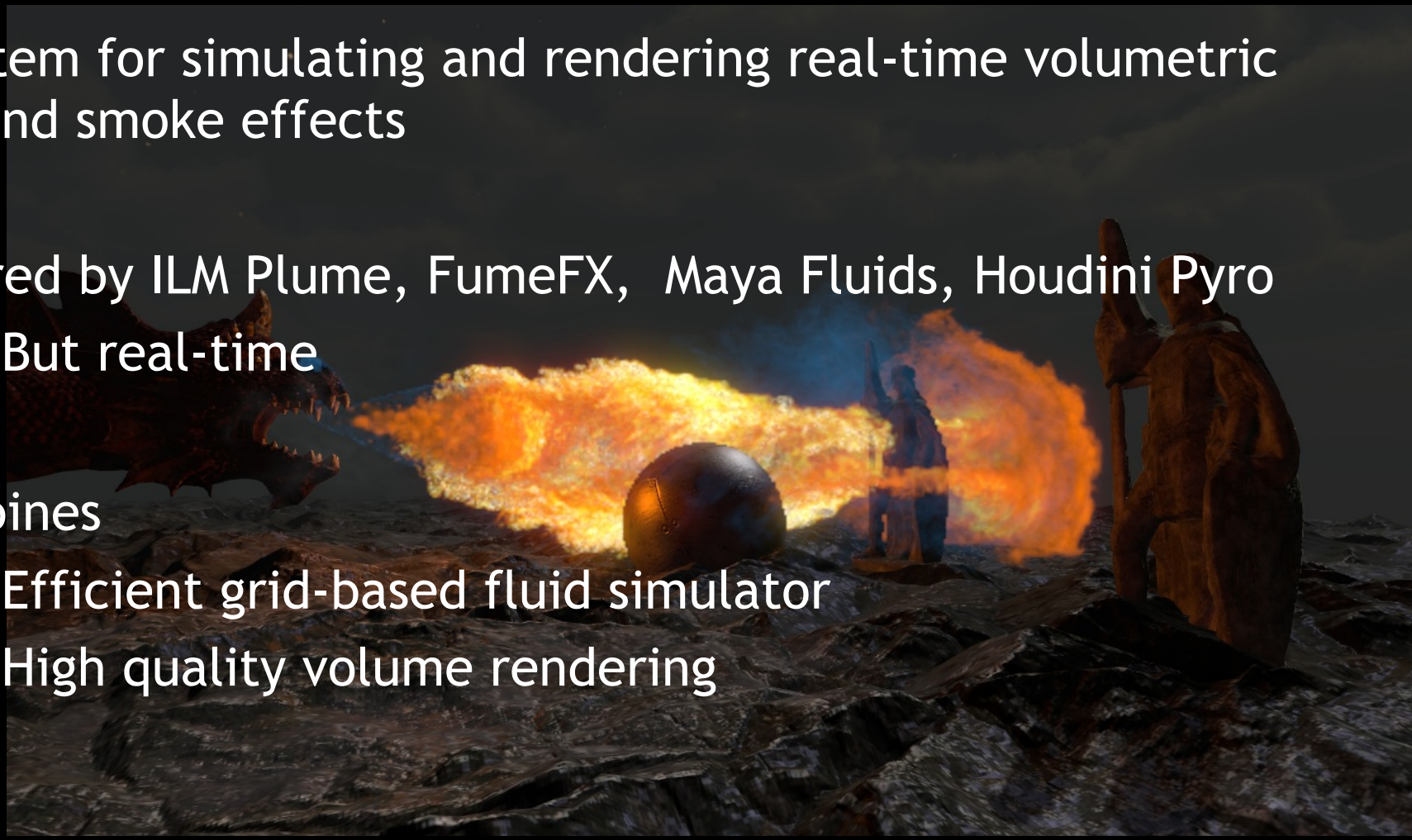
UE4 upcoming

FLAMEWORKS

- A system for simulating and rendering real-time volumetric fire and smoke effects

- Inspired by ILM Plume, FumeFX, Maya Fluids, Houdini Pyro
  - But real-time

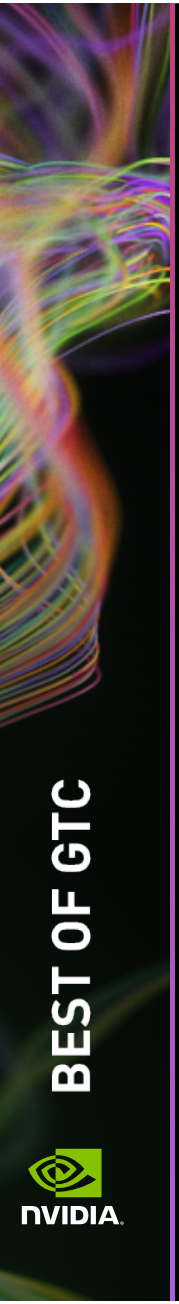
- Combines
  - Efficient grid-based fluid simulator
  - High quality volume rendering



BEST OF GTC

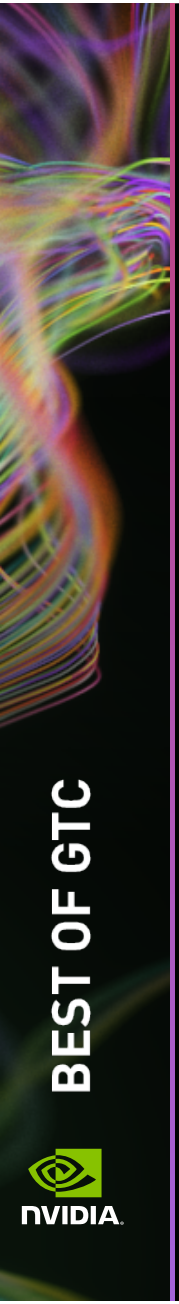
NVIDIA

FIRE IN CURRENT GAMES



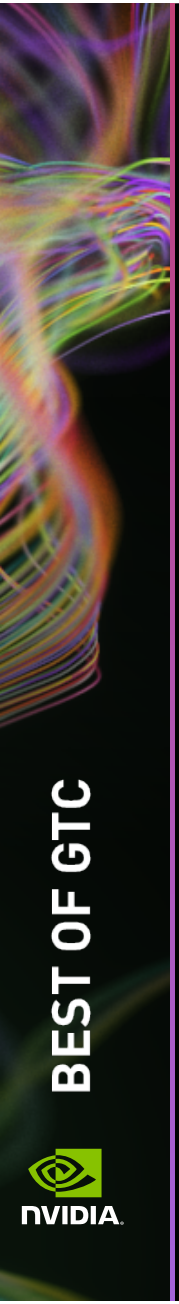
ADVANTAGES OF FLAMEWORKS

- Non-repeating effects
- Interactive
- Volume rendering avoids particle “cotton ball” look
- Less memory?
- Higher quality?



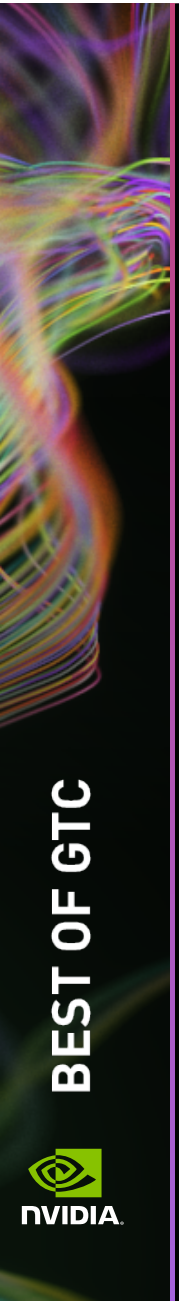
GOALS

- High quality
- Fast, efficient, scalable
- Simple, easy to integrate into games
- Customizable



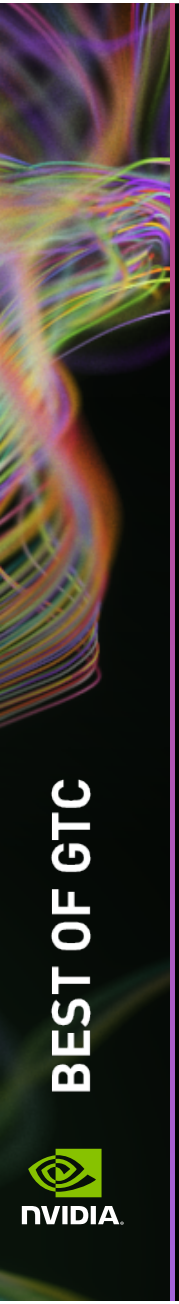
IMPLEMENTATION DETAILS

- Implemented as library
  - Simple API
  - Can be called from tools or game engine
  - Working on integration into UE4 and other engines
- Current implementation uses DirectX 11 (compute shaders)
  - best fit for games
  - good graphics interoperability
  - potential for simultaneous graphics-compute
- OpenGL version possible in future
  - Performance of mobile GPUs is increasing quickly



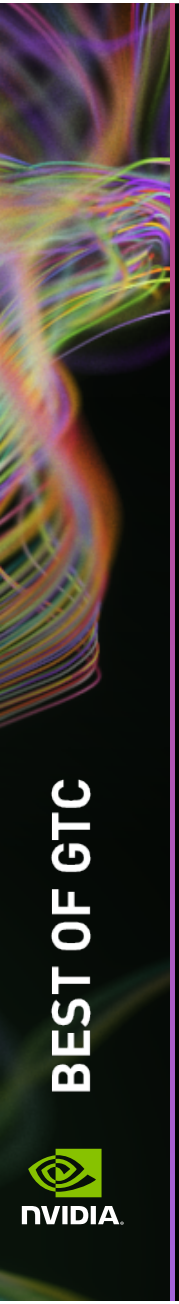
DRAGON DEMO

- 256 x 128 x 128 velocity grid (4M voxels)
- 2x density res multiplier = 512 x 256 x 256 (32M voxels)
- ~30fps on GeForce Titan



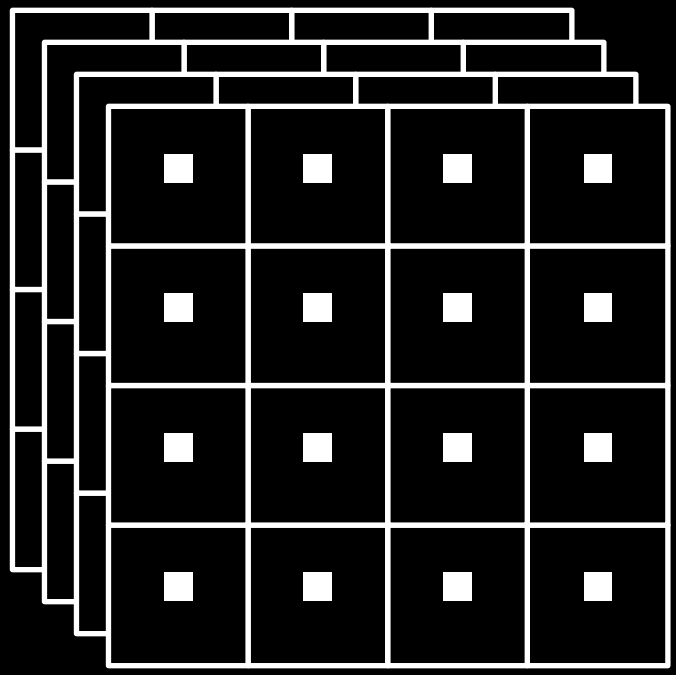
SIMULATION

- Grid-based fluid simulator
  - Gas simulator really - no liquid surface tracking etc.
  - No particles
- Features
  - Multi-grid solver
  - MacCormack advection (second order accurate)
  - Vorticity confinement
  - Combustion model
  - Density resolution multiplier (upscaling)

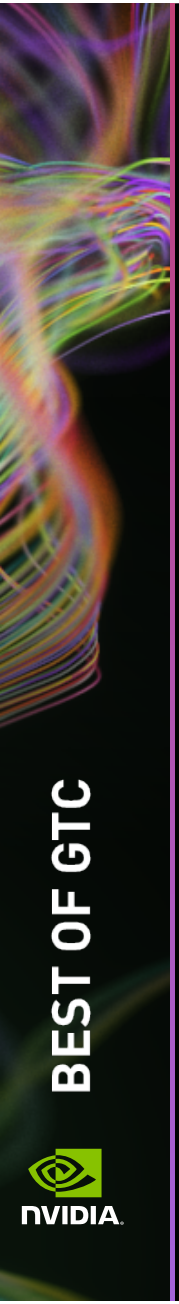


SIMULATION

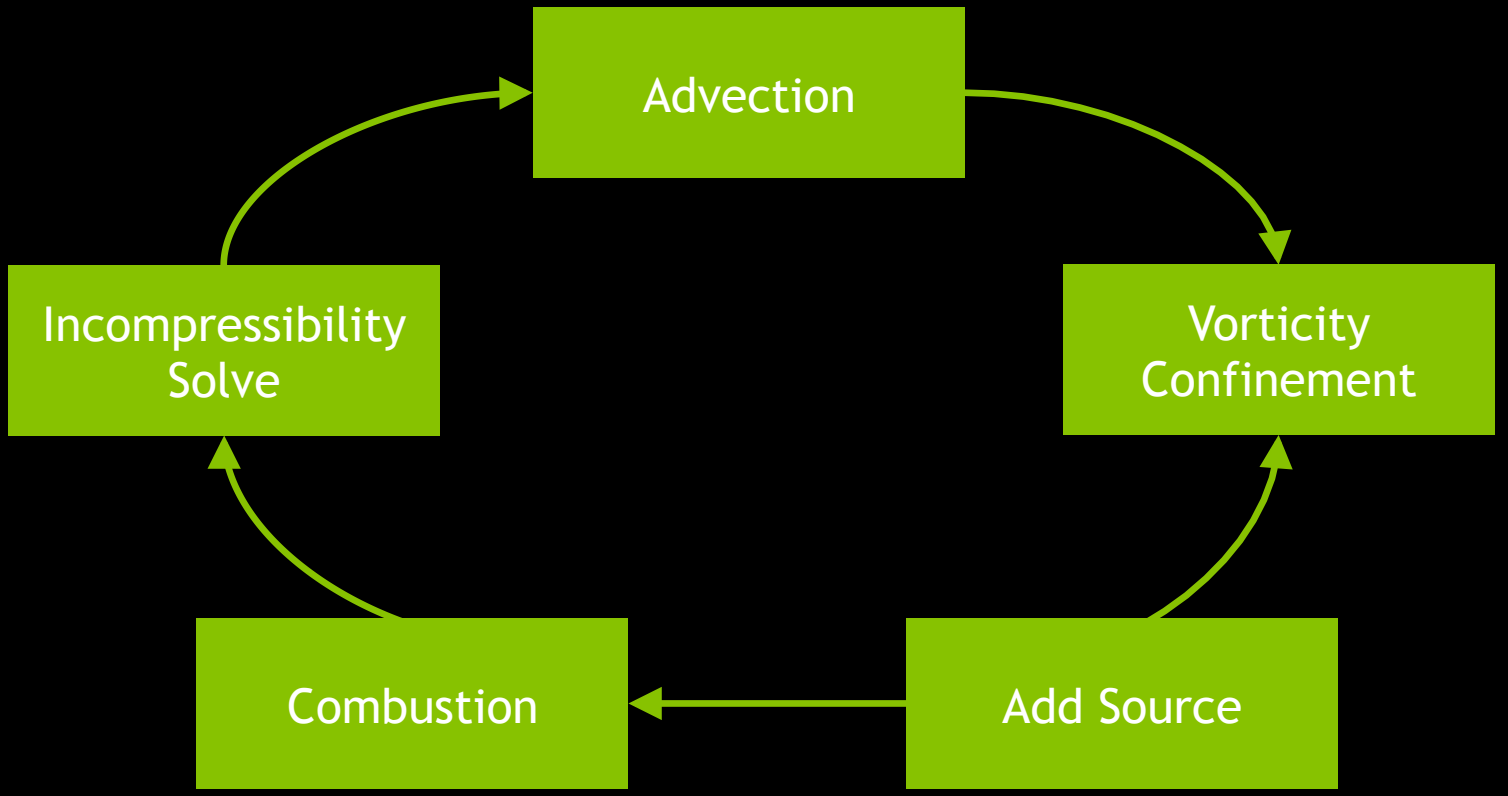
- Quantities
  - Velocity
  - Temperature
  - Fuel
  - Smoke Density

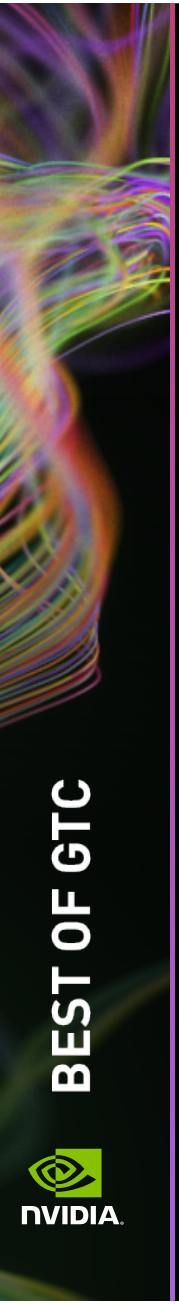


- Discretization
  - Use collocated grid i.e. all quantities stored at cell center
  - Stored in 3D textures
  - Half (fp16) and float precision options

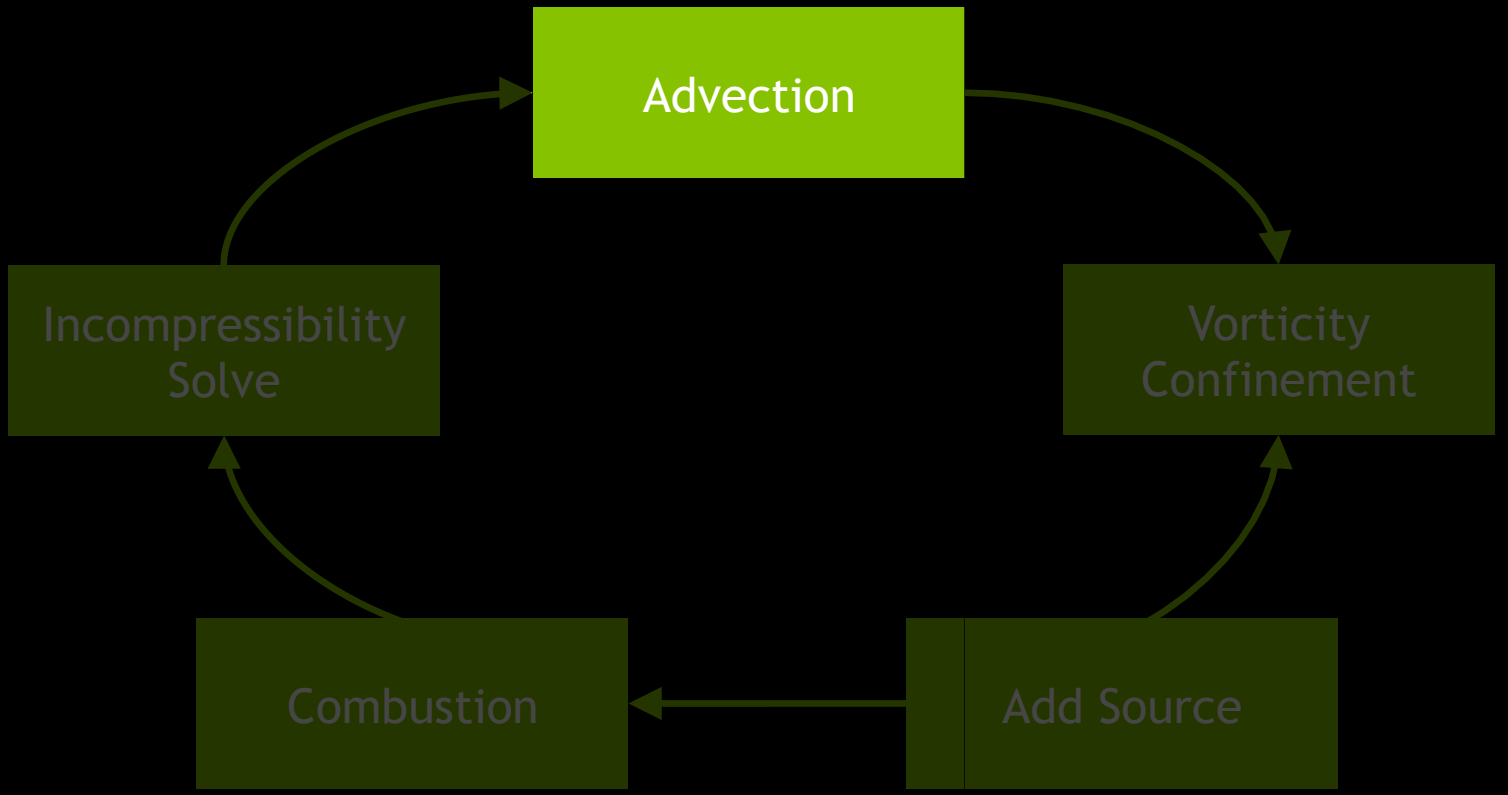


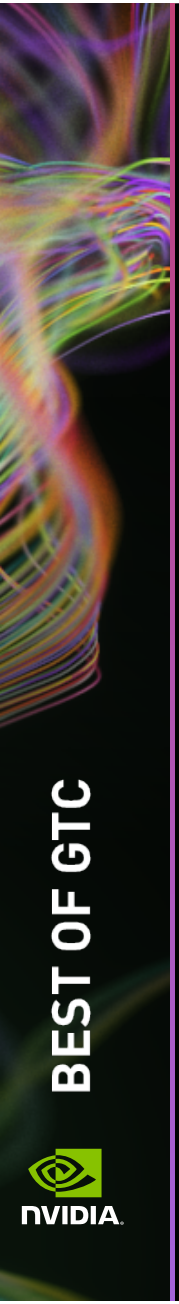
SIMULATION LOOP





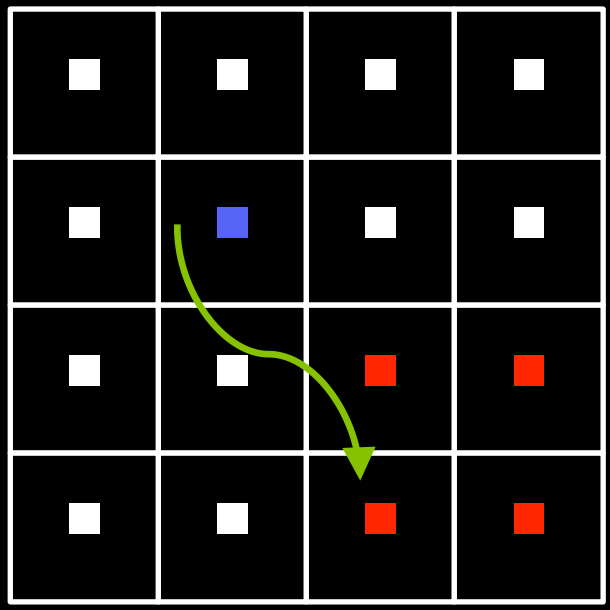
SIMULATION LOOP

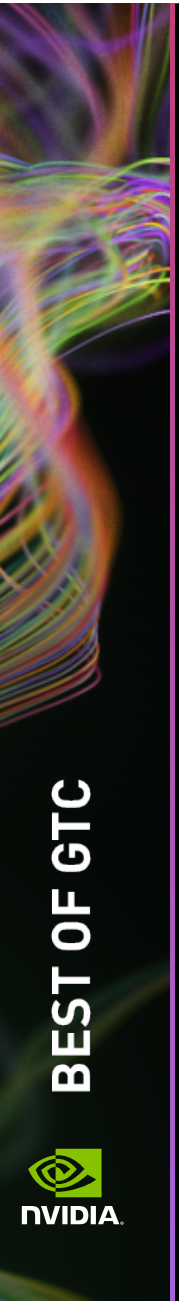




ADVECTION

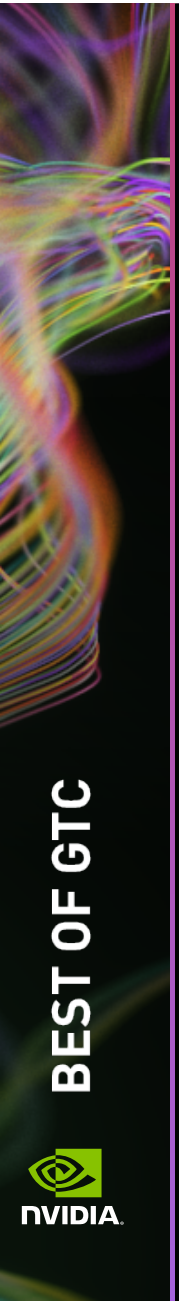
- Fluid physical quantities such as temperature, density etc. are moved by fluid velocity
  - Fluid velocity is also moved by fluid velocity!
- Semi-Lagrangian Method [Stam 99]
  - Start at a cell center
  - Trace velocity field backward in time
  - Interpolate quantity from grid
    - Tri-Linear interpolation
- Optionally, move fuel at faster speed





ADVECTION

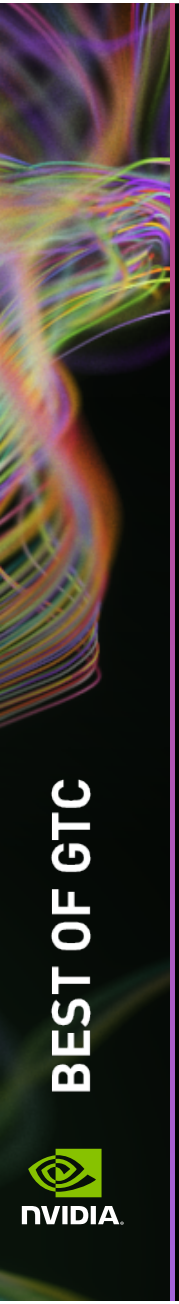
- Semi-Lagrangian method is first order accurate
  - Causes a lot of numerical diffusion
  - Smoke and flames smooth out
  - Small scale details disappear
  - Vortices disappear
- Use Modified MacCormack Method [Selle et al. 08]
  - Second order accurate
  - Tracing backward and then forward in time to estimate the error
  - Subtract the estimated error off to get a more accurate answer



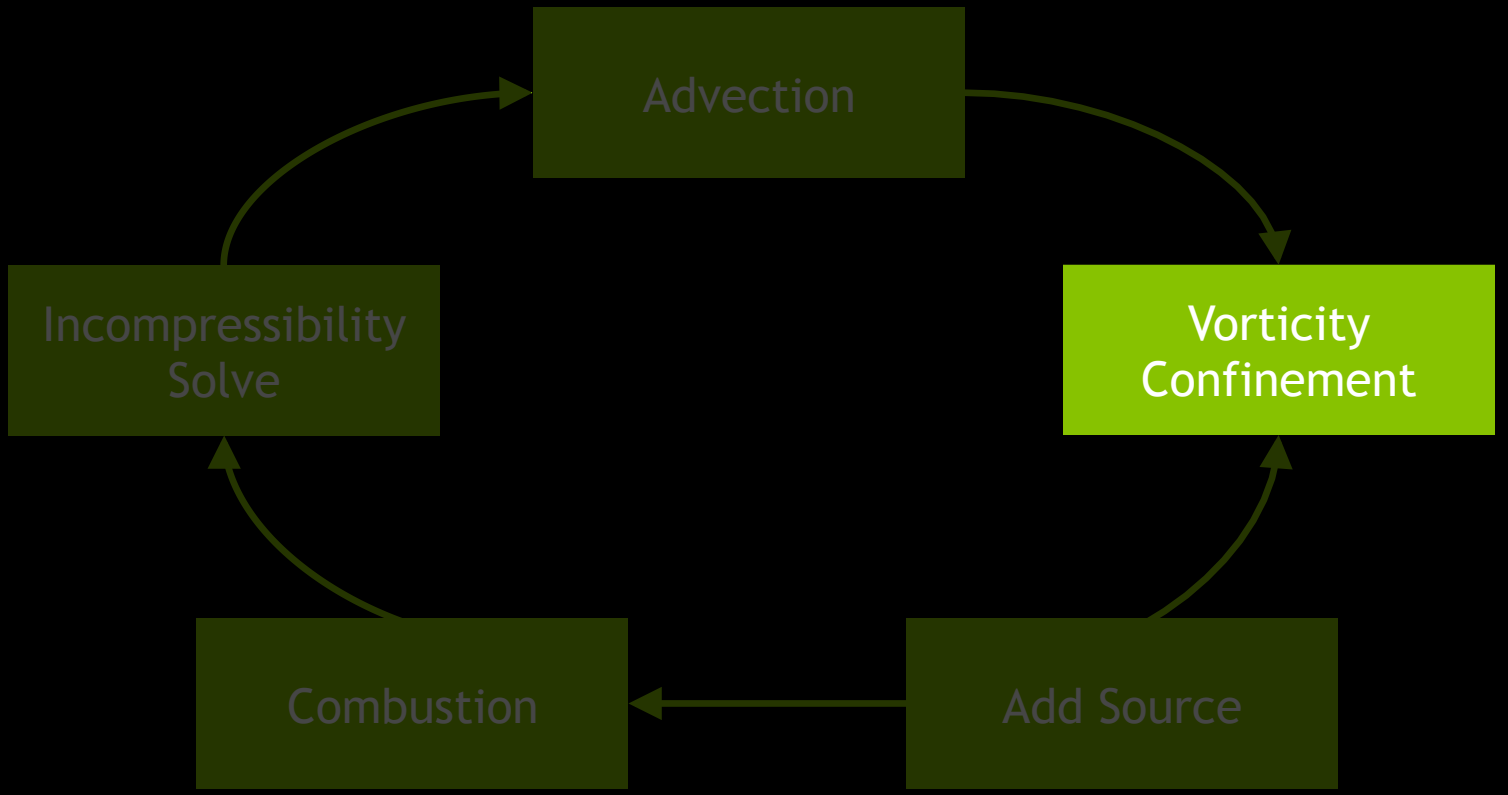
ADVECTION

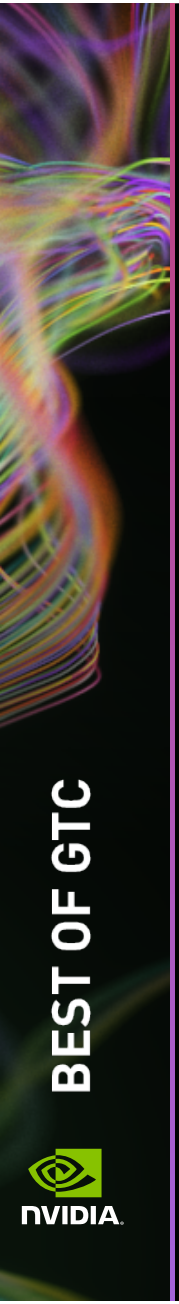
Semi-Lagrangian

MacCormack



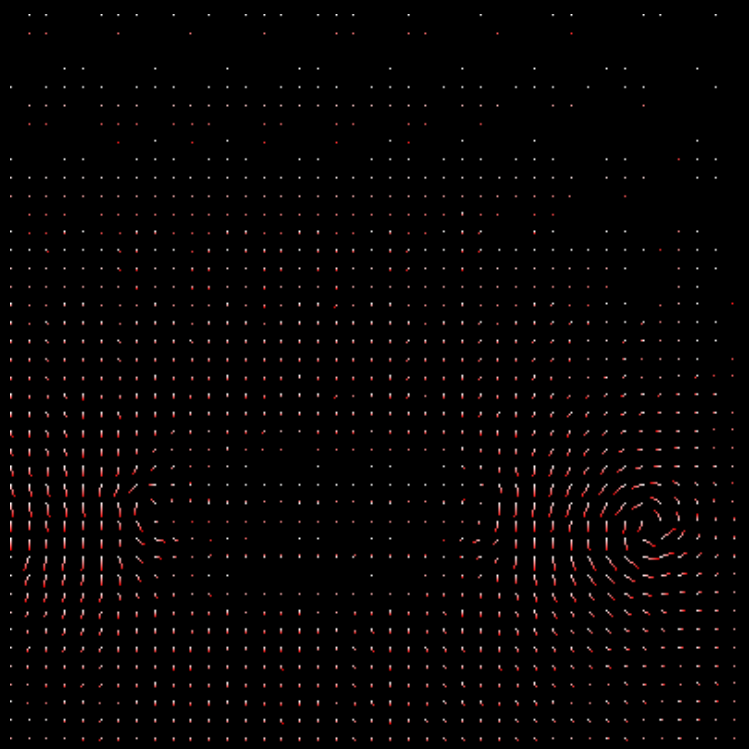
SIMULATION LOOP



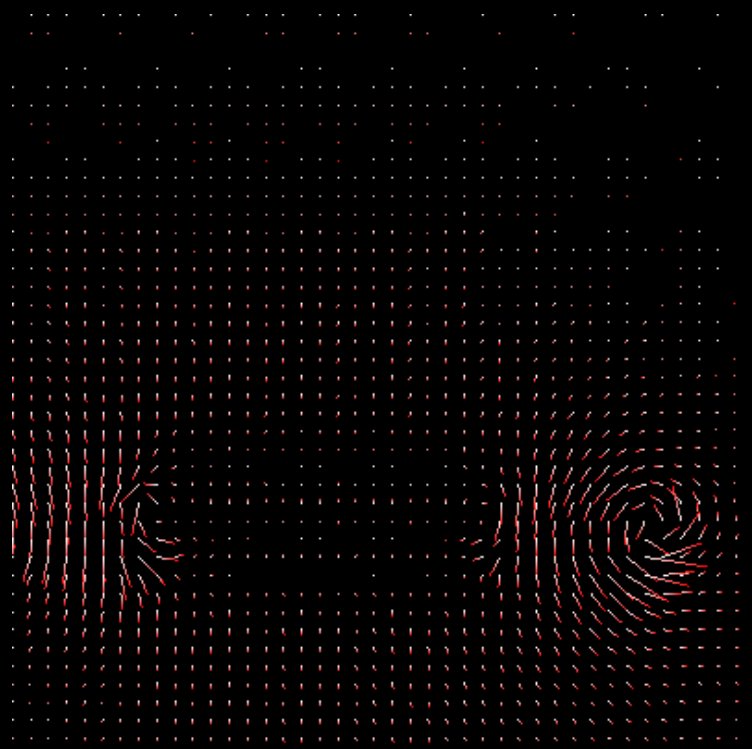


VORTICITY CONFINEMENT

- Identify where vortices are, then add force to amplify them [Fedkiw et al. 01]

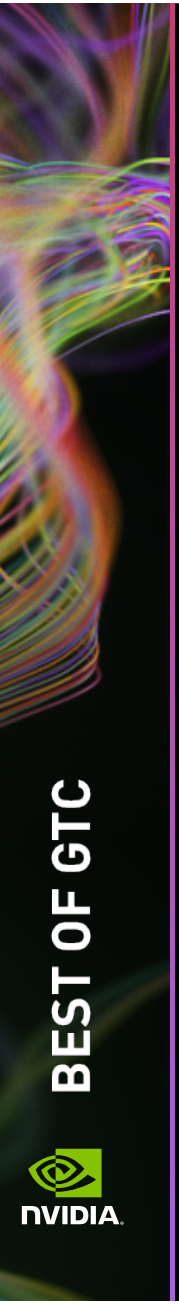


Before



After*

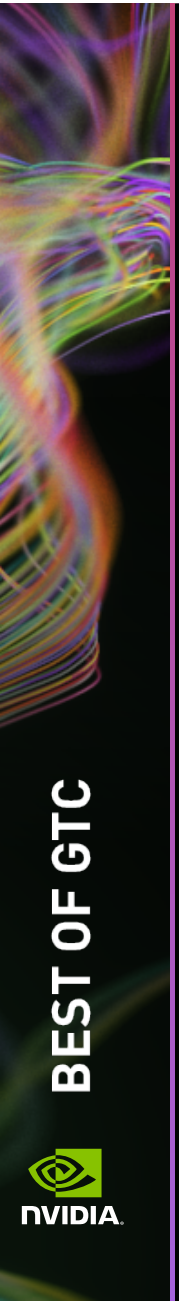
*exaggerated for visualization purpose



VORTICITY CONFINEMENT

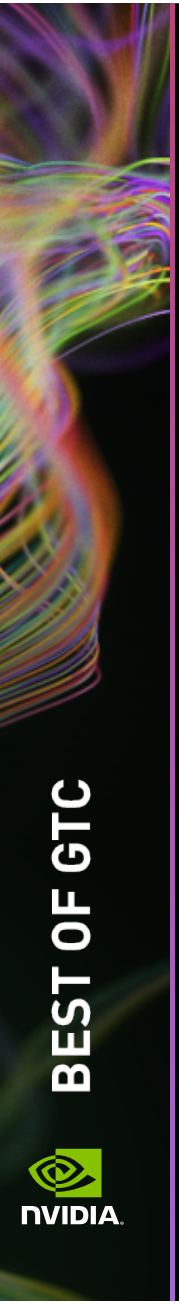
Without vorticity confinement

With vorticity confinement

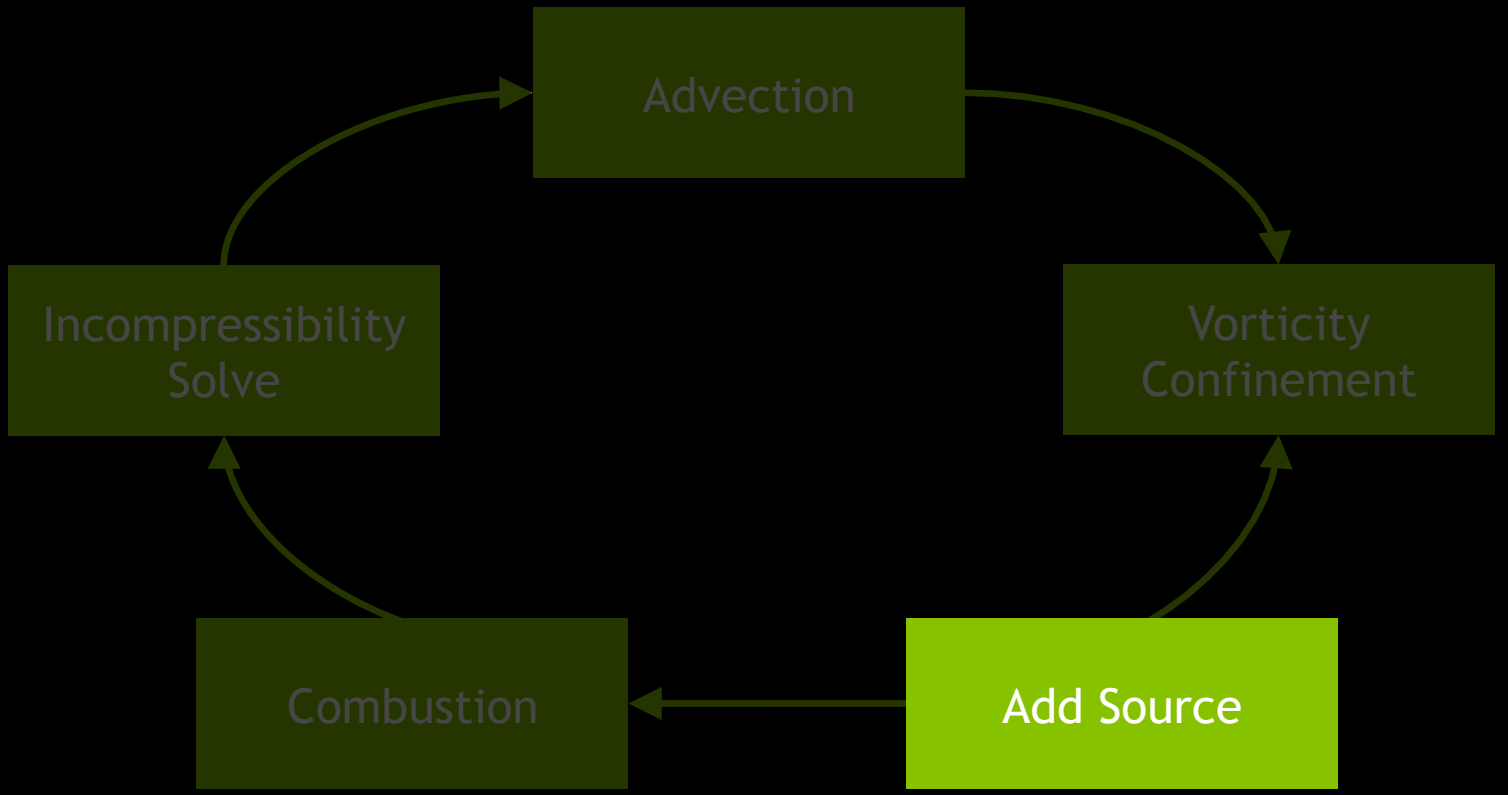


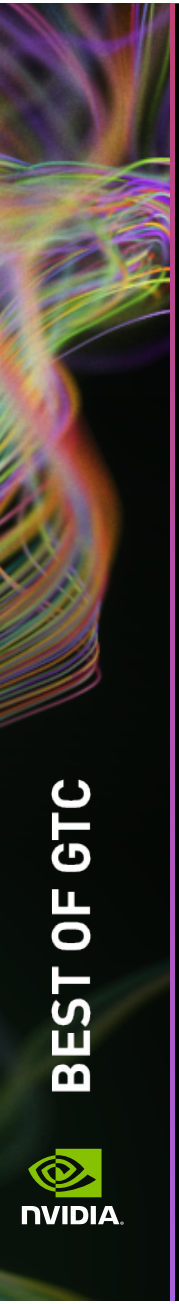
ADDITIONAL FORCES

- Procedural noise
- Buoyancy
- Custom force fields



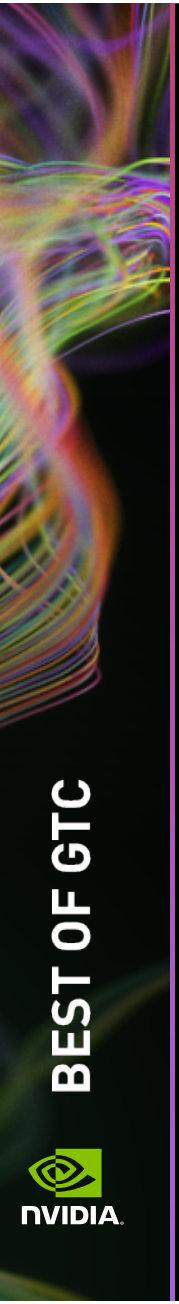
SIMULATION LOOP





EMITTERS

- Add density, temperature, fuel and velocity to simulation
- Currently supported shapes
  - Sphere
  - Plane
  - Box
  - Cone
- Custom emitters possible using callbacks
  - Developer can write HLSL compute shaders that write to density and velocity textures



CUSTOM EMITTER HLSL EXAMPLE

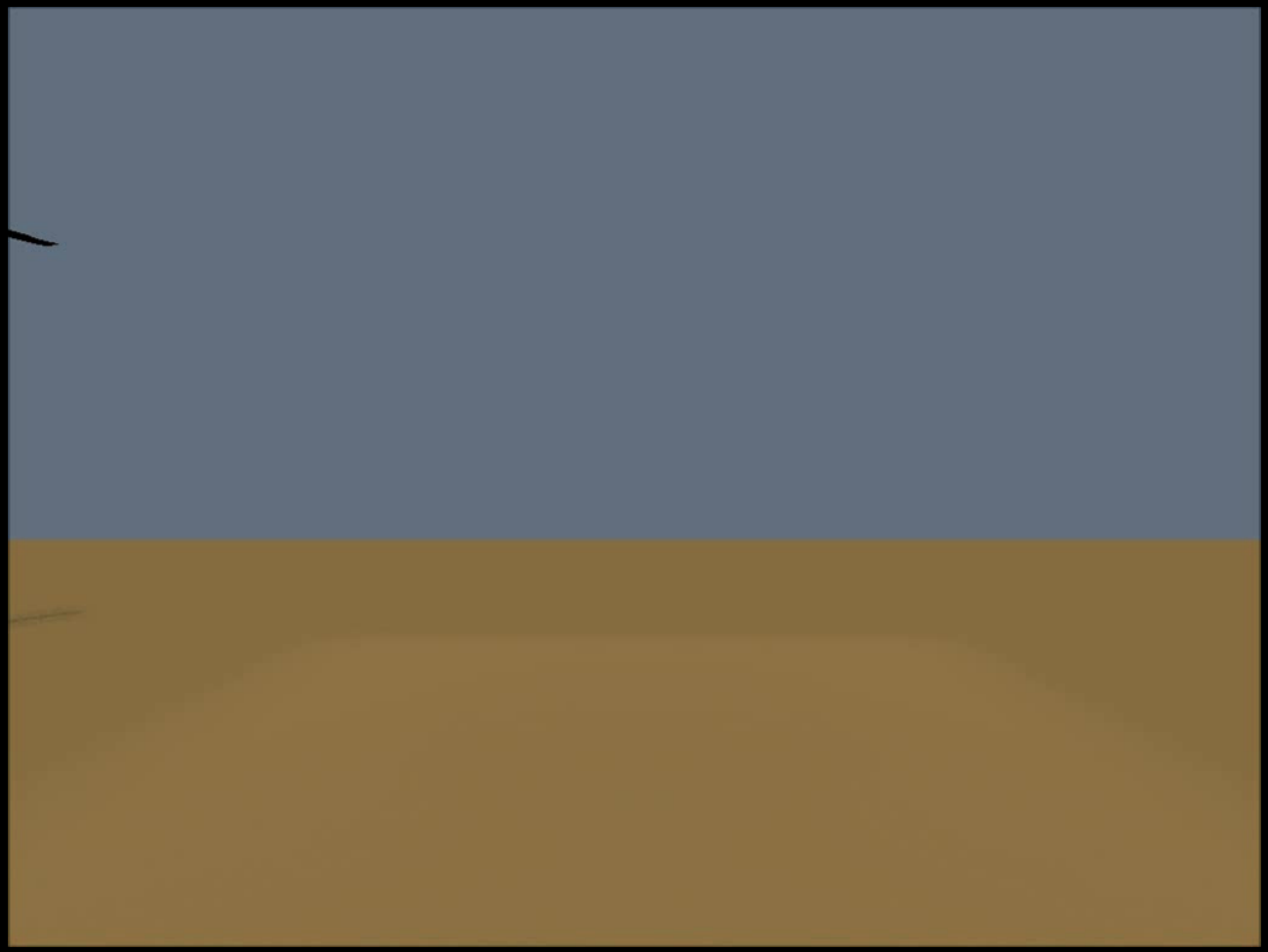
```
Texture3D<float4>   srcTex : register (t0);
RWTexture3D<float4> dstTex : register (u0);

[numthreads(THREADS_X, THREADS_Y, THREADS_Z)]
void DensityEmitter(uint3 i : SV_DispatchThreadID)
{
    // read existing data at this cell
    float4 d = srcTex[i];

    float3 p = voxelToWorld(i);
    float r= length(p - emitterPos);
    if (r <  emitterRadius) {
       d.x += smoothstep(emitterRadius, emitterInnerRadius, r);
    };

    // write new data
    dstTex[i] = d;
}
```

HELICOPTER LANDING DEMO

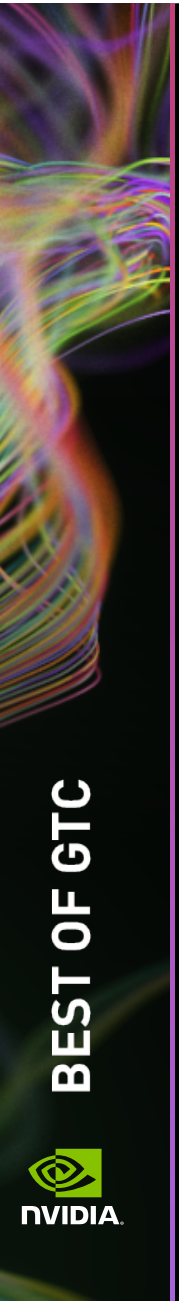


BEST OF GTC

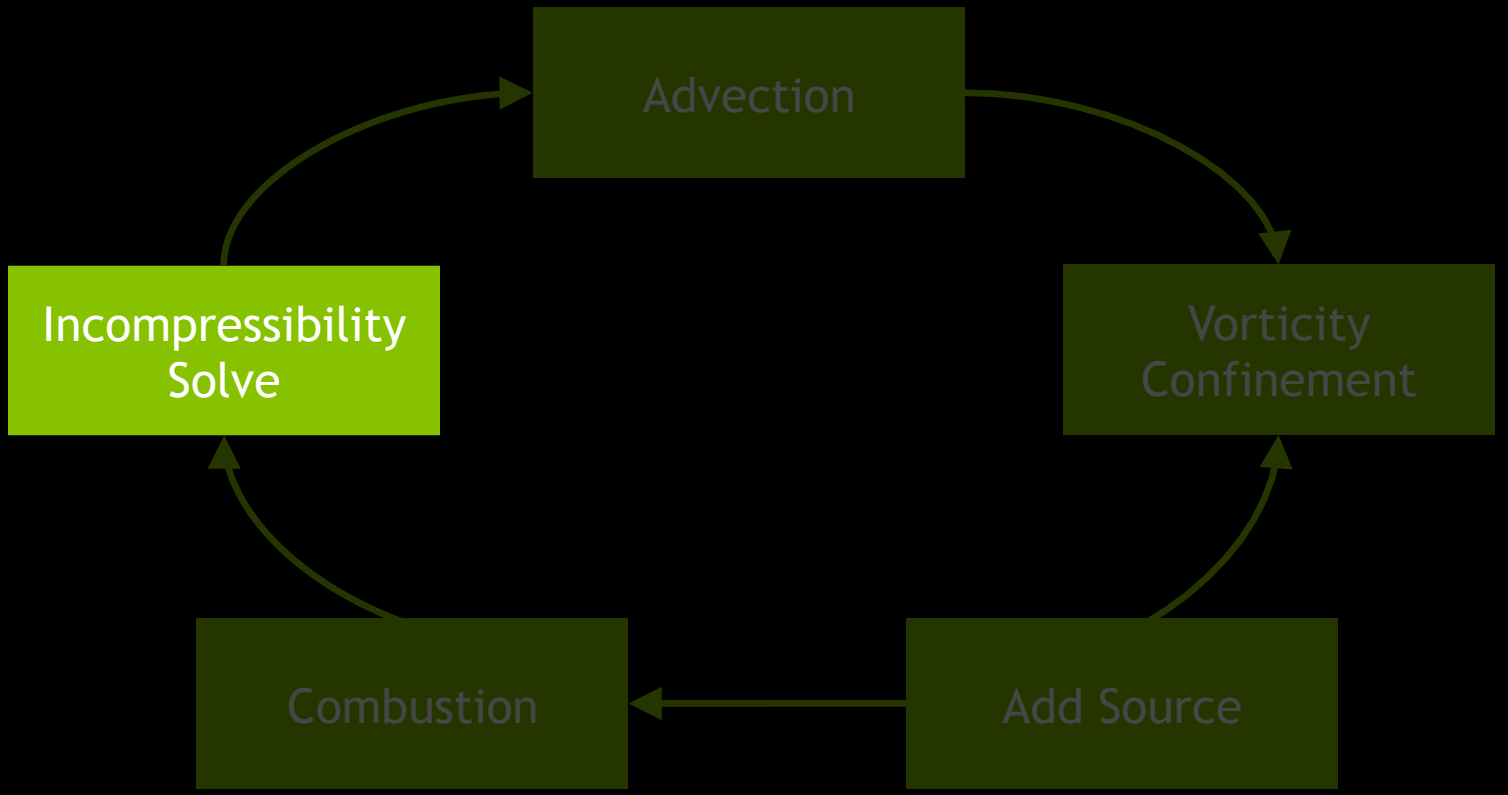
NVIDIA

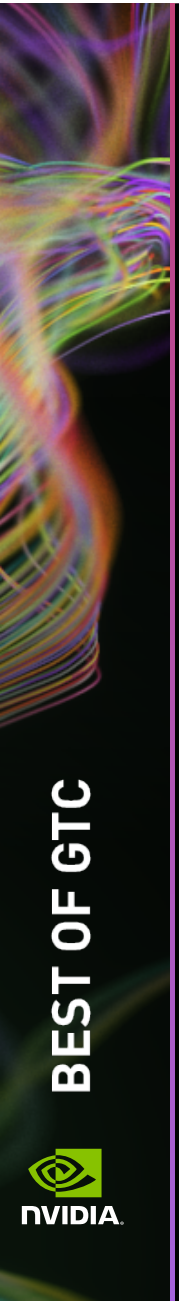
COMBUSTION MODEL

- If temperature is above ignition temp and there is fuel, combustion occurs:
  - Consumes fuel
  - Increases temperature
  - Generates expansion by modifying simulation divergence [Feldman & O'Brien 03]
  - Temperature causes upwards buoyancy



SIMULATION LOOP



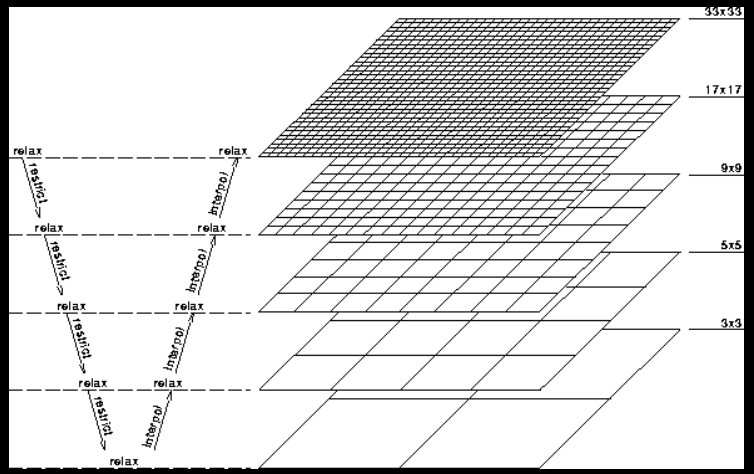


INCOMPRESSIBILITY SOLVER

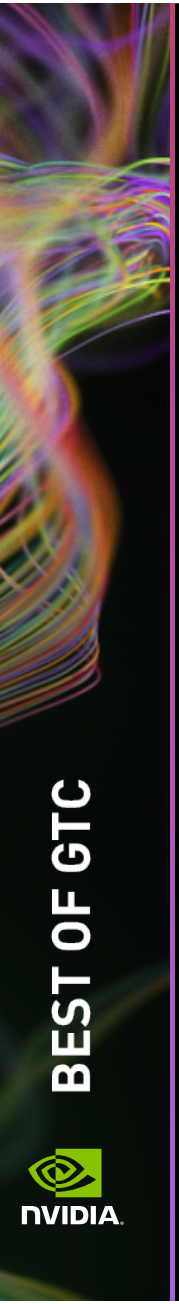
- Goal: Make velocity field divergence free
- Why?
  - Incompressible fluid -> divergence free velocity field
  - Responsible for swirling motion commonly associated with fluid
  - Conserves mass
- How?
  - Compute a pressure field whose gradient canceled out divergence of velocity field [Stam 99]
  - Pressure field is the solution to a linear system
  - We use a geometric multigrid solver

SOLVING LINEAR SYSTEM

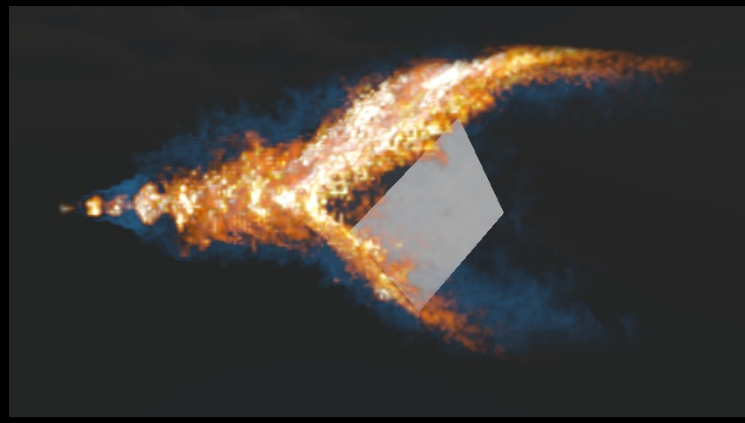
- Geometric multi-grid
  - Open boundary on top, sides
  - Closed boundary on bottom
  - Ignores internal obstacles



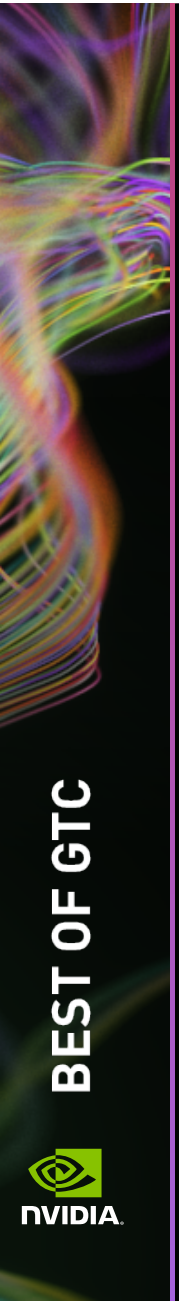
- Use Iterate Orthogonal Projection (IOP) [Molemaker et al. 08], to enforce solid boundary condition for internal obstacles
  - Set the normal component of fluid velocity to that of solid
  - Do multi-grid
  - Set the normal component of fluid velocity to that of solid
  - (can repeat for more accurate solution)



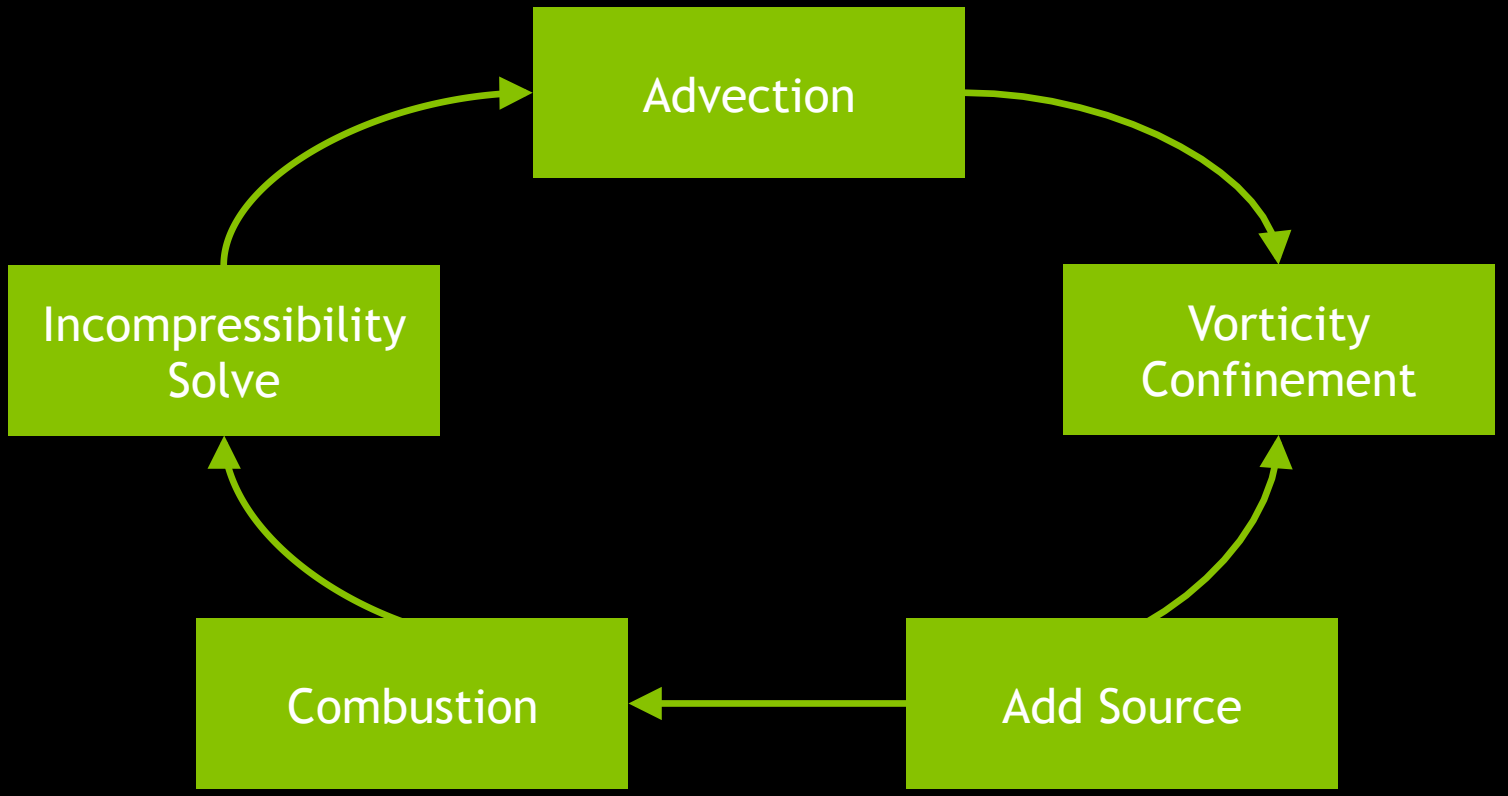
OBSTACLES

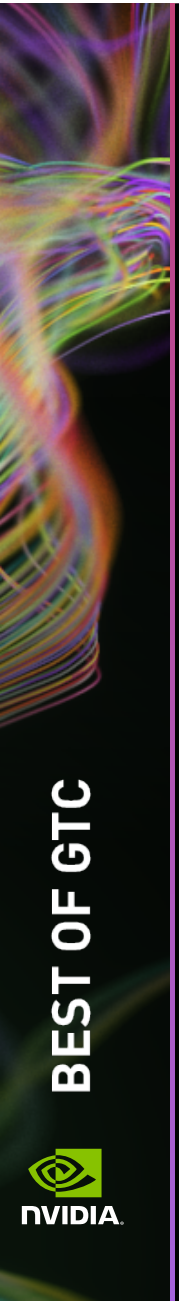


- Currently support:
  - Implicits: sphere, capsule, box
  - Pre-computed signed distance fields (stored on a grid)
    - Can be scaled, translated, rotated during run time
- We rasterize the following to grid at each time step:
  - Distance to nearest obstacle
  - Normal and velocity of the nearest obstacle
- Solver then use these fields for all simulation steps
  - Advection and incompressibility solve



SIMULATION LOOP

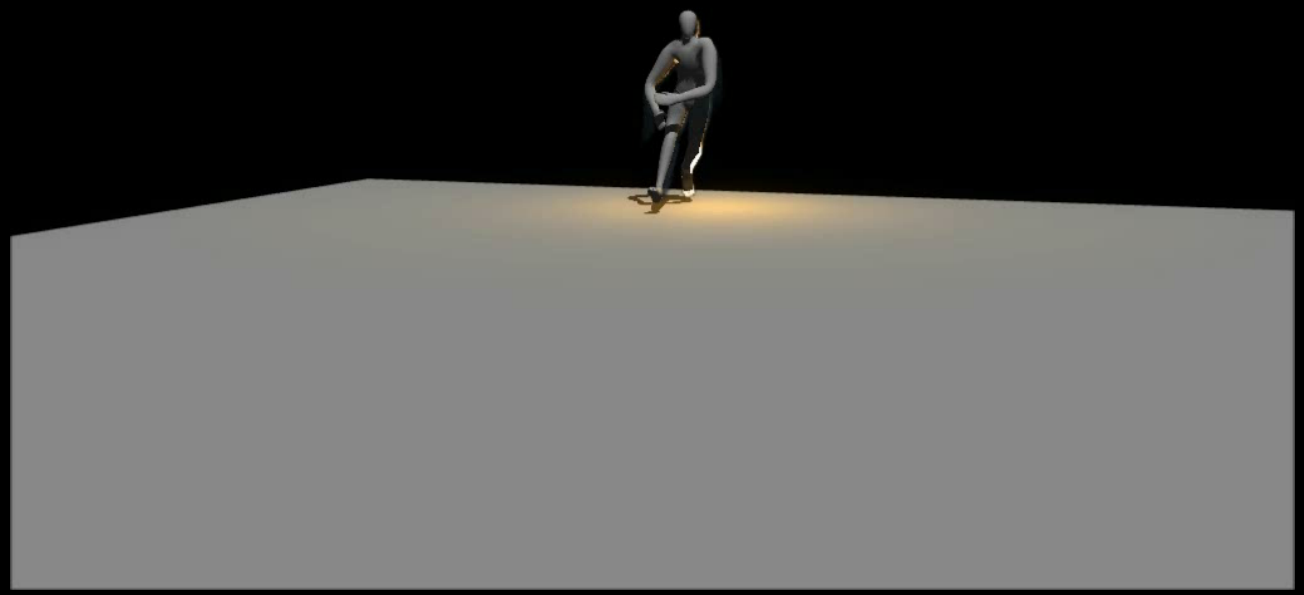




MOVING GRID

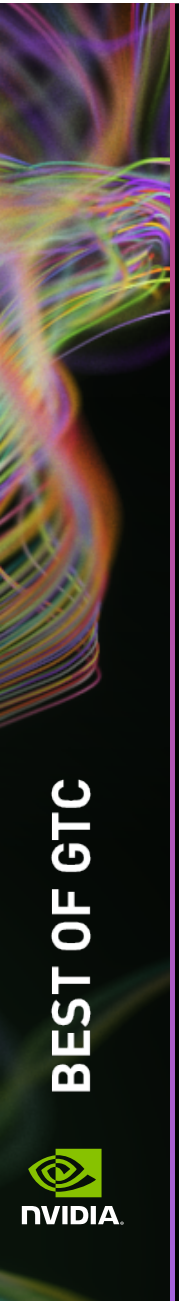
- Grid is fixed size
  - Easy to hit the sides
- Grid can be translated to follow moving objects
  - Implemented by basically adding opposite velocity during advection
- Grid is always axis-aligned
  - May support grid rotations in future version

DANCER DEMO



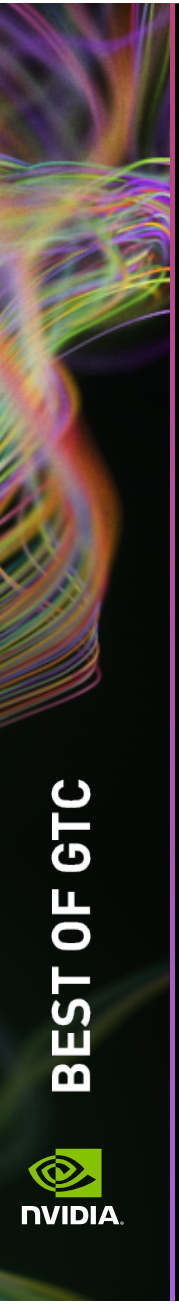
BEST OF GTC

NVIDIA

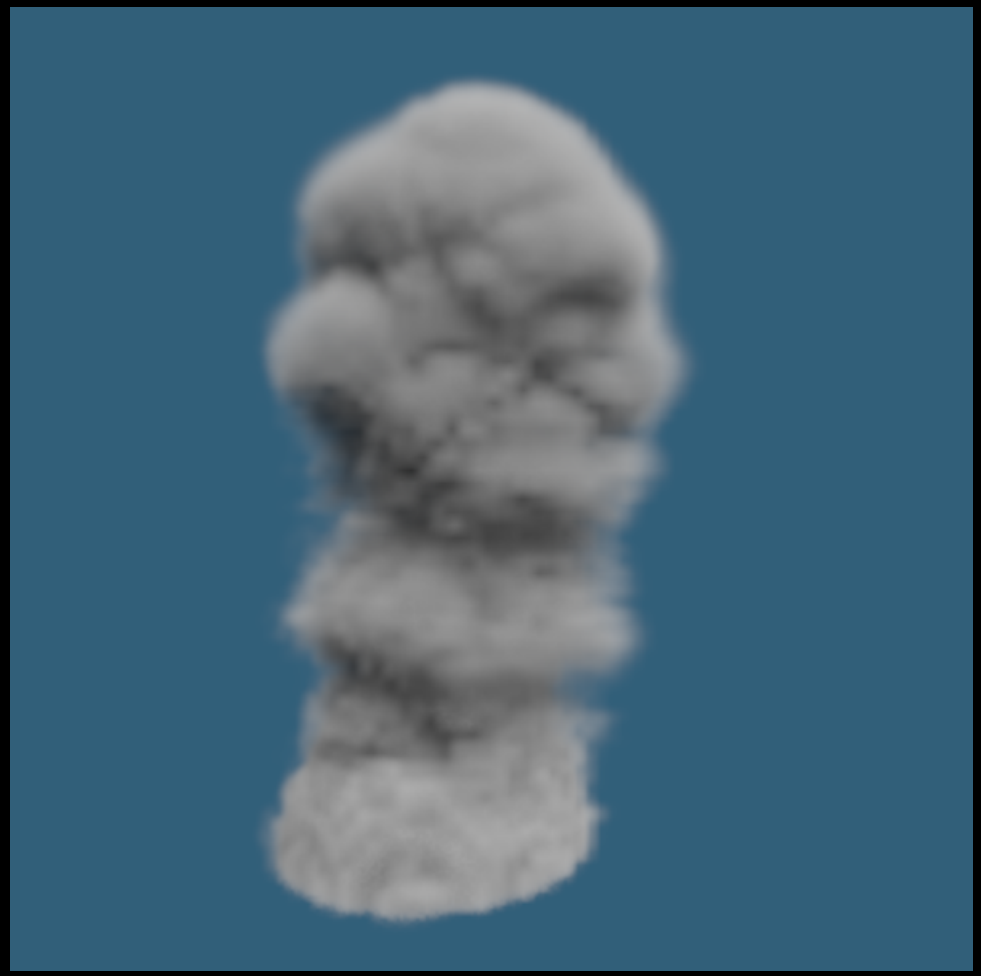


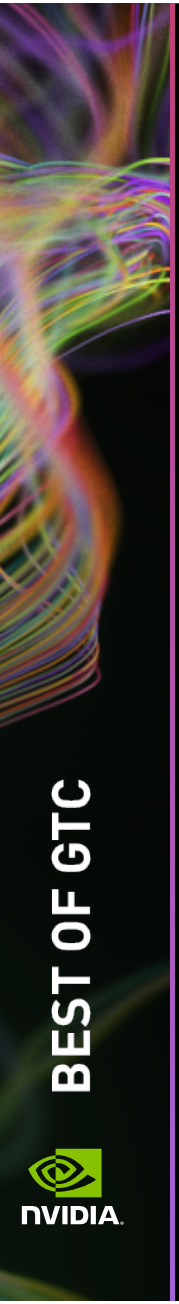
HIGH-RES DENSITY / UPSCALING

- Density grid can be stored at higher resolution than velocity
  - Usually 2x or 4x resolution
- Improves performance
  - velocity solver is main bottleneck
- Velocities are interpolated during density advection
  - Not strictly correct since interpolated velocities aren't divergence-free
  - But looks fine

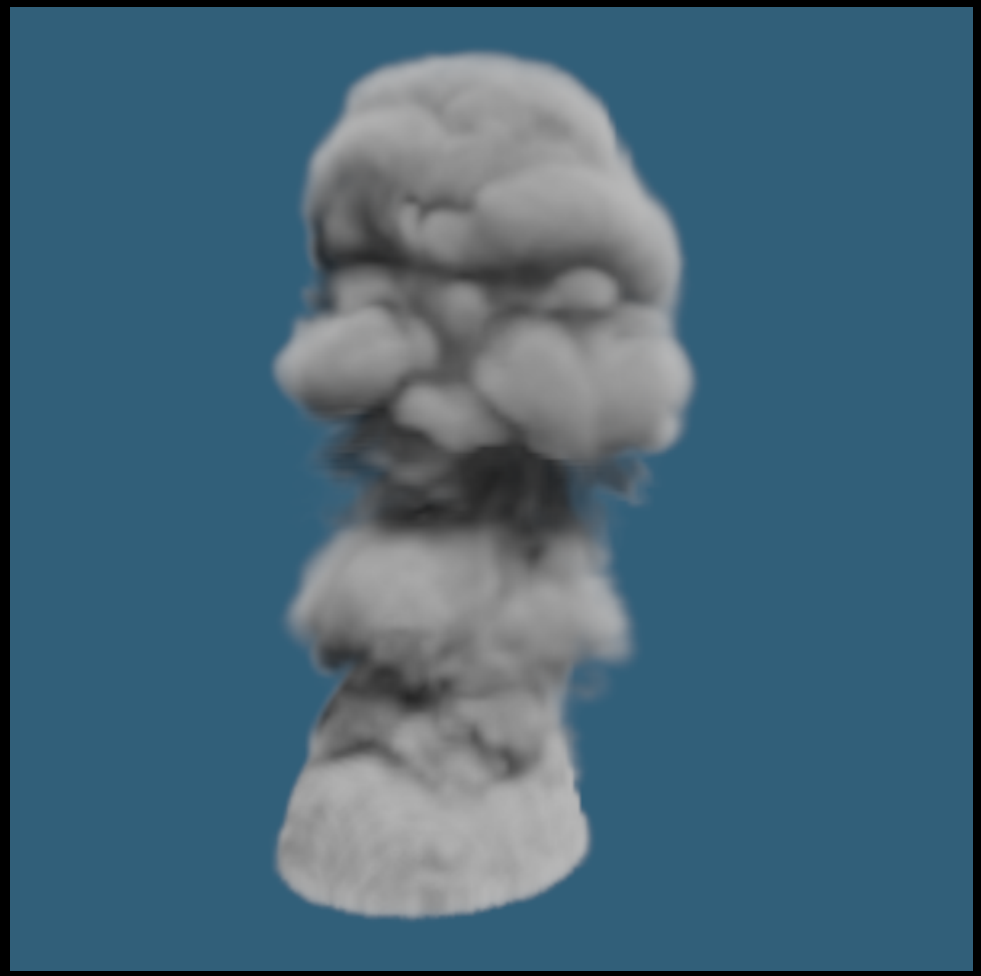


DENSITY 1X

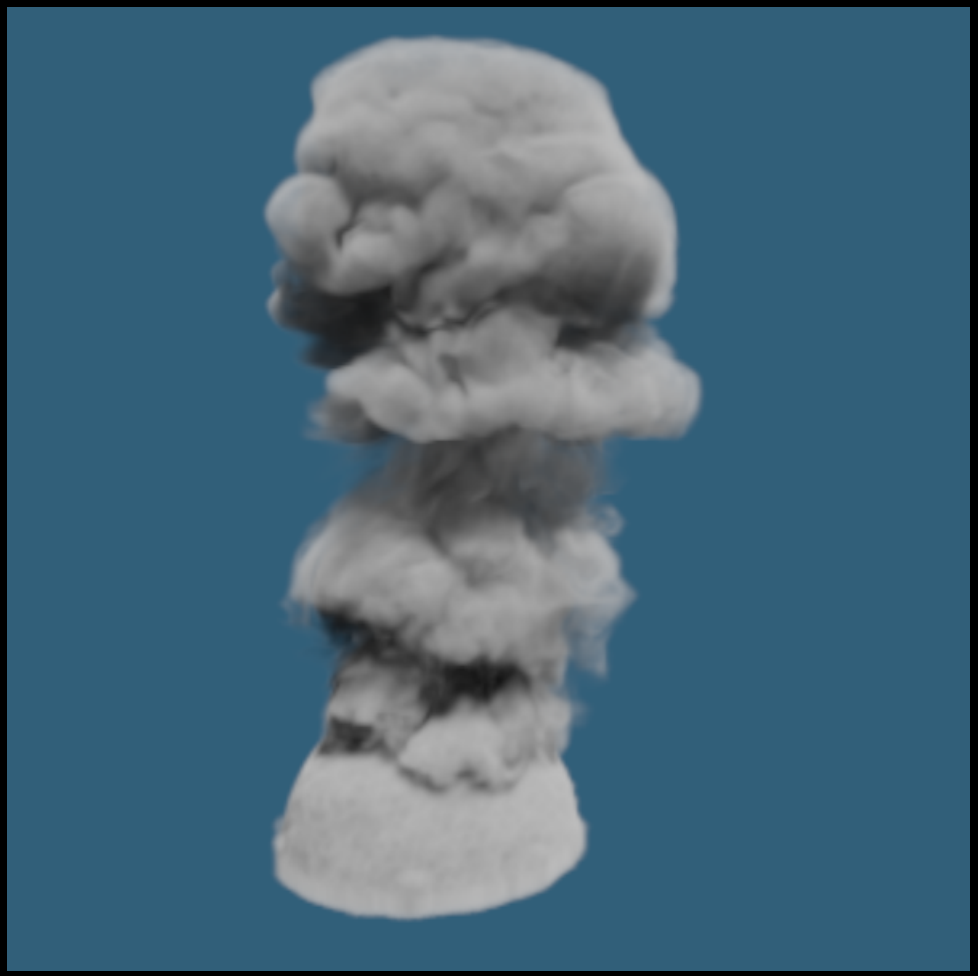


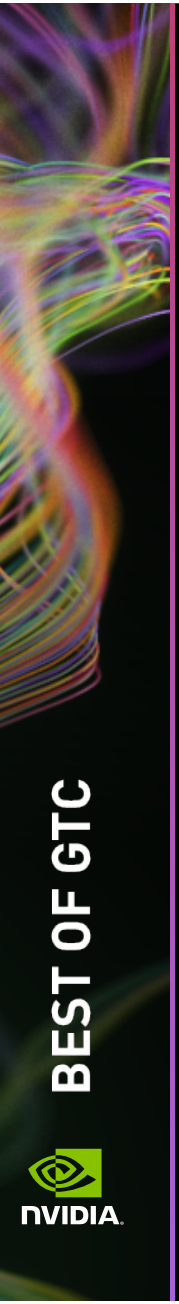


DENSITY 2X



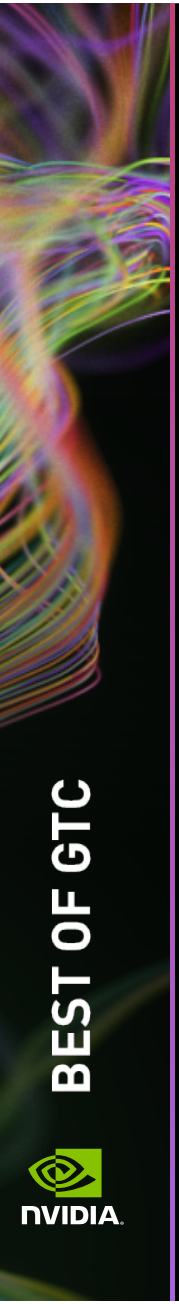
DENSITY 4X





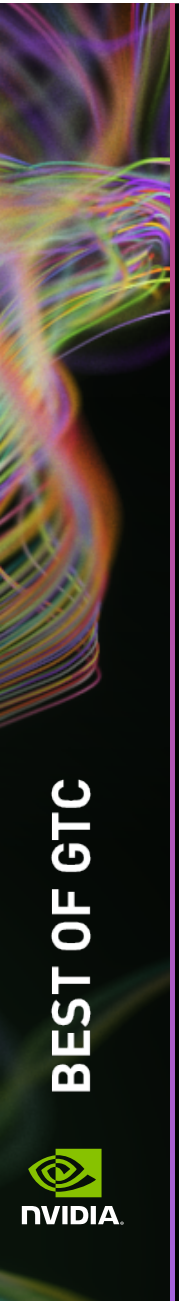
RENDERING OPTIONS

- Particles
  - Advect particles through velocity field from simulation
  - Render as point sprites
  - Requires depth sorting for correct bending
- Iso-surface rendering
  - Using marching cubes or ray marching to extract surface
- Volume Rendering
  - Highest quality option



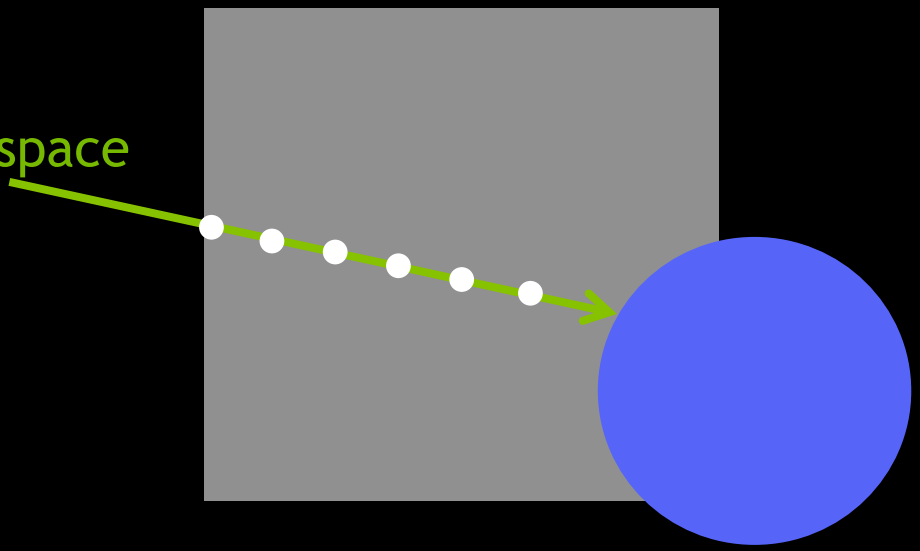
VOLUME RENDERING

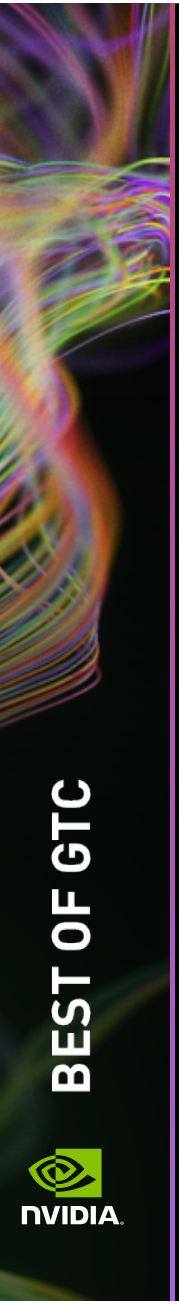
- Separate part of FlameWorks
  - Optional - developers are free to do their own rendering
- Features
  - Color mapping
  - Depth compositing
  - Edge fade
  - Heat Haze



VOLUME RENDERING

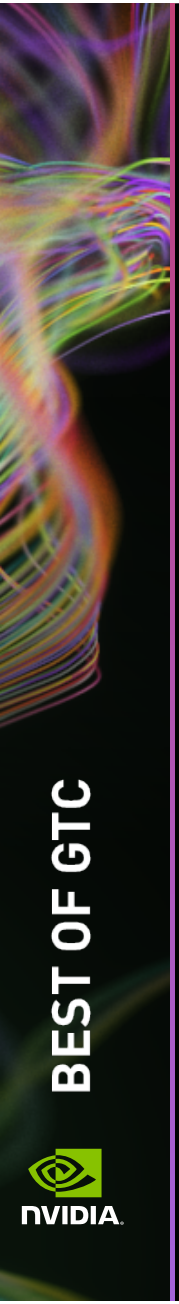
- Implemented using ray-marching in pixel shader
- Can render at low resolution (½ or ¼) and then up-sample
- Intersect ray against bounding box to get entry/exit points
- Ray march from front to back
  - Allows early exit if volume becomes opaque
- Empty space skipping
  - volumes often have a lot of empty space

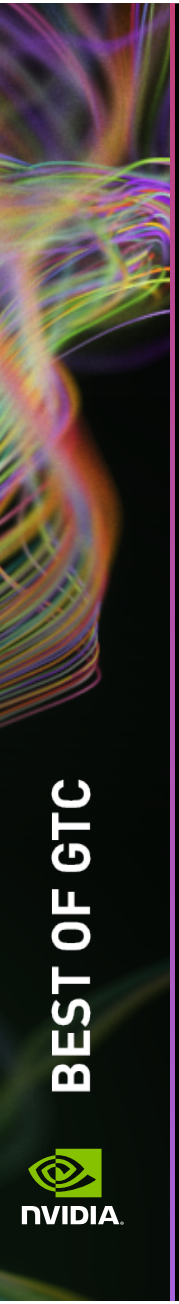


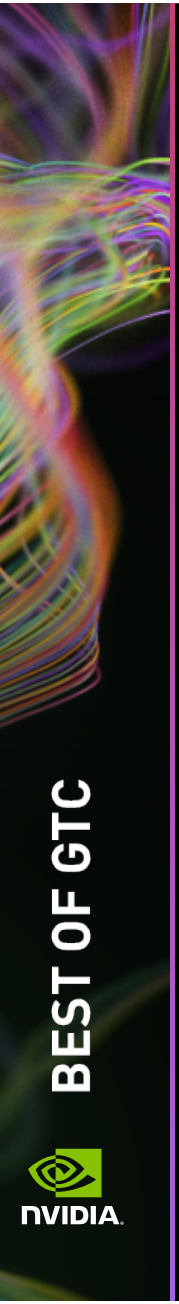


COLOR MAPPING

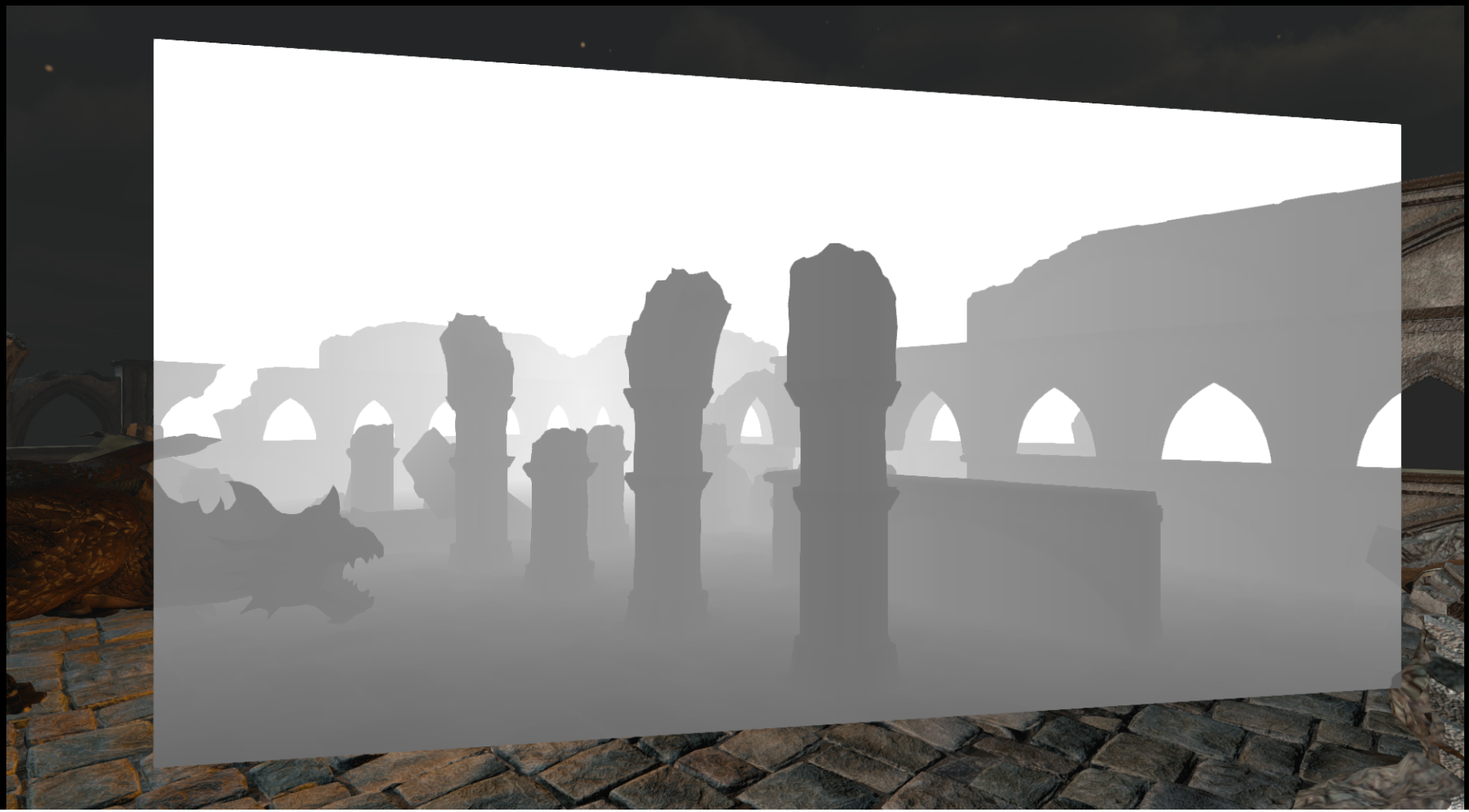
- Temperature is mapped to color using 1D texture LUT
  - Can also effect alpha (transparency)
- Can use a physically-based Blackbody radiation model
  - (see GTC 2012 talk)
- Or arbitrary artist-defined map


BEST OF GTC
NVIDIA


BEST OF GTC
NVIDIA

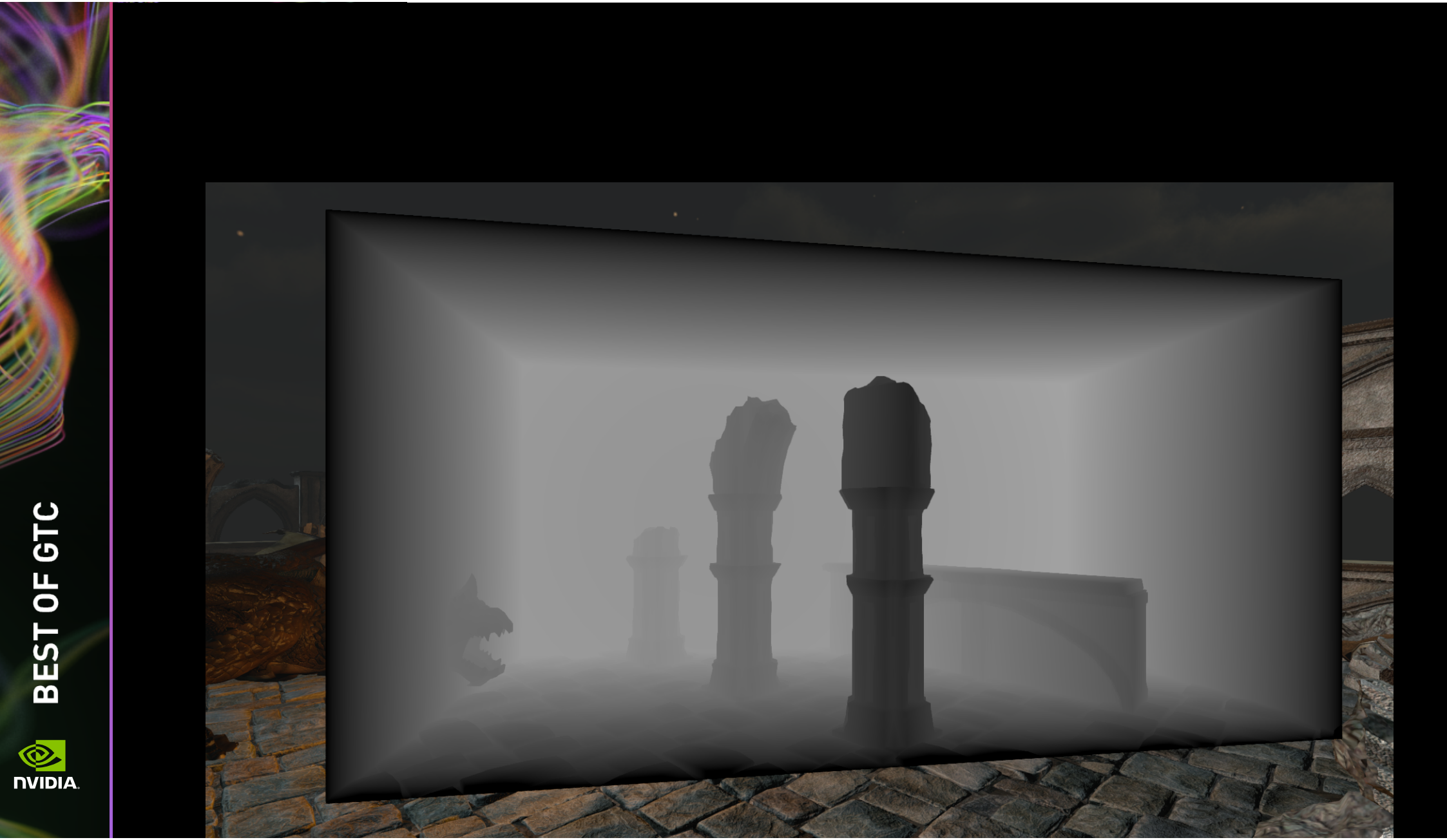

BEST OF GTC
NVIDIA

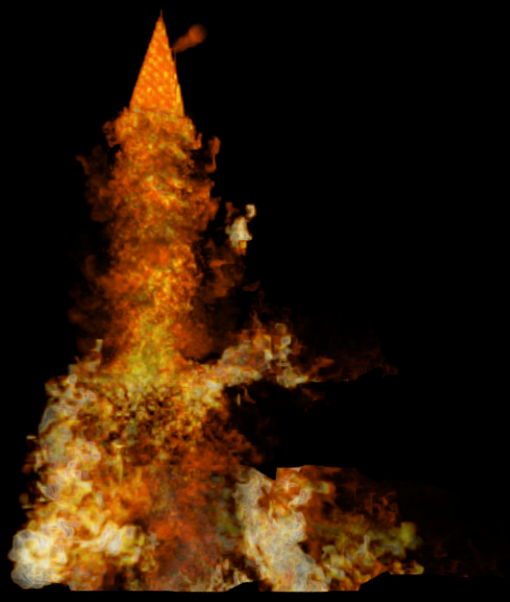
COMPOSITING

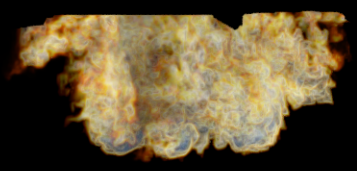


BEST OF GTC

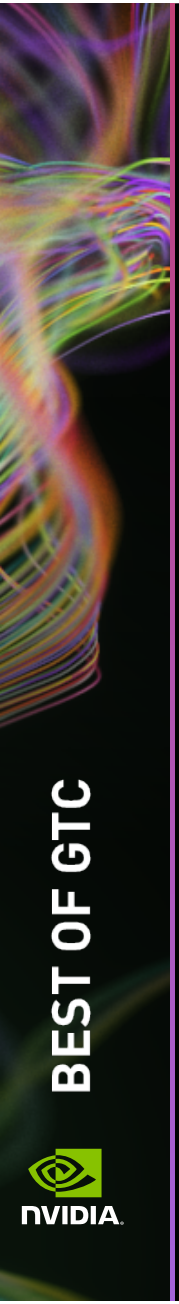
NVIDIA


BEST OF GTC
NVIDIA






BEST OF GTC
NVIDIA



HEAT HAZE

- Adds to perception of heat
- Offset background lookup based on gradient of temperature (or just luminance) in screen space

HEAT HAZE - OFF

BEST OF GTC

NVIDIA

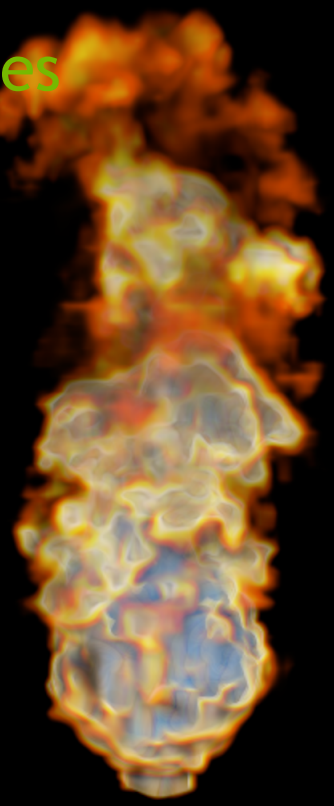
HEAT HAZE - ON

BEST OF GTC

NVIDIA

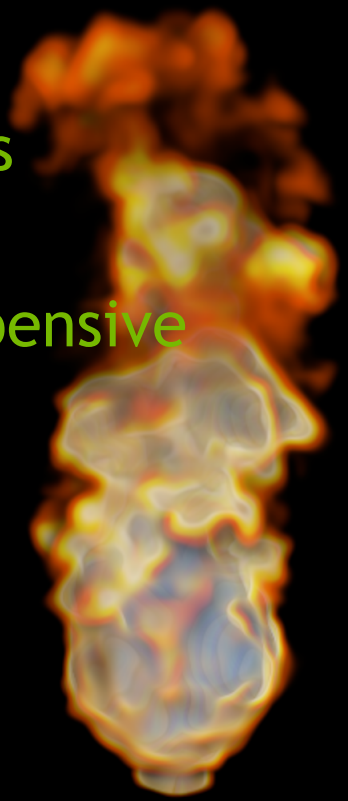
LINEAR FILTERING

- Fastest option
- Some artifacts at low res

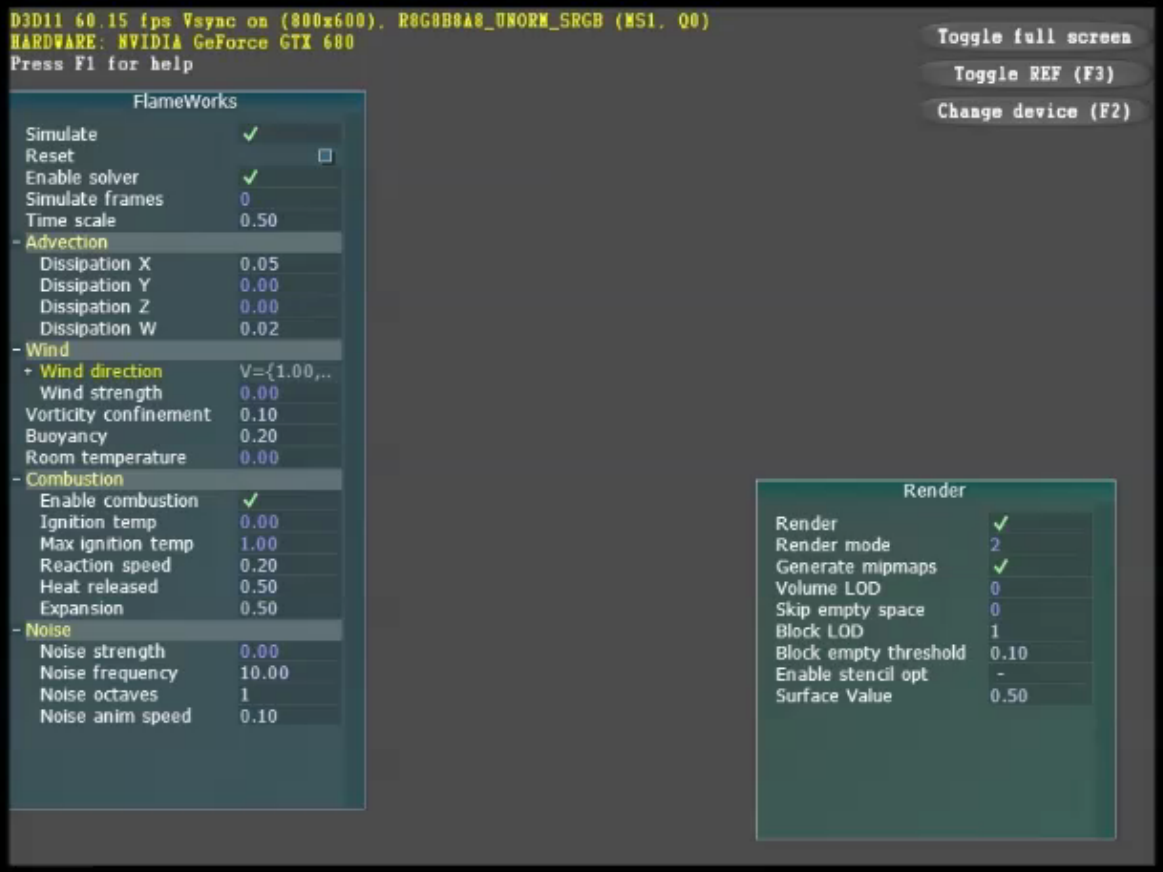


CUBIC FILTERING

- Smoother
  - Blurs some details
- Uses 8 bilinear lookups
  - [Sigg & Hadwiger 05]
- But ~8 times more expensive

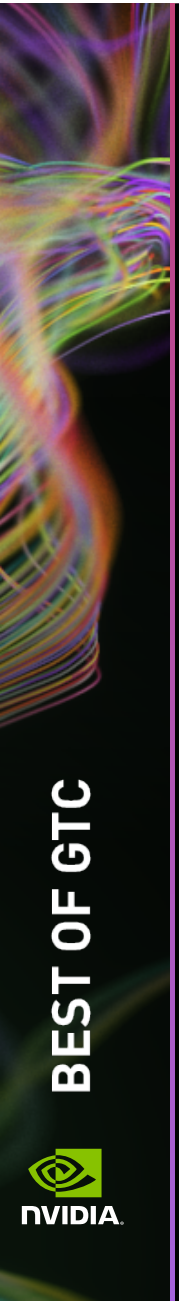


ALTERNATIVE LOOKS - TOON SHADING



BEST OF GTC

NVIDIA



VOLUMETRIC SHADOWS

- Shadows are very important for smoke, dust etc.
- We generate a dense 3D shadow volume as a pre-pass
- Can be generated at fraction of density texture resolution
  - (½ or ¼ typically)
- Ray march towards light, computing transmittance
- For dense volumes, only need small number of samples (2-8)
- Problem: banding visible
- Solution: jitter ray start position towards light by random amount
- Causes noise, so blur result in 3D

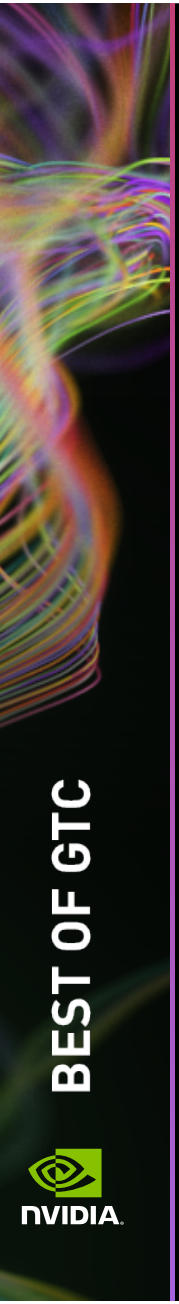
SHADOWS - OFF

SHADOWS – 2 SAMPLES

SHADOWS - 4 SAMPLES

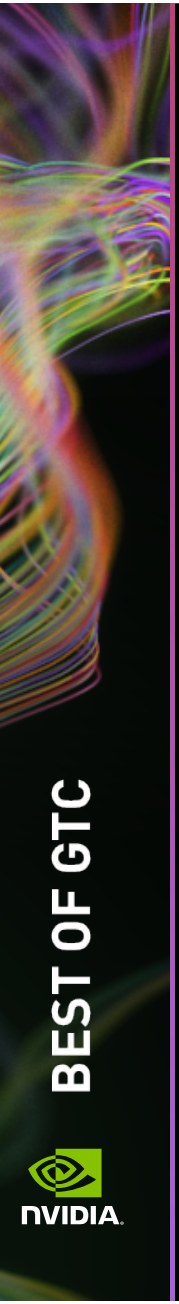
SHADOWS - 4 SAMPLES + JITTER

SHADOWS - 4 SAMPLES + JITTER + BLUR



VOLUME SCATTERING APPROXIMATION

- Scattering is an important effect for smoky volumes
- Simple approximation:
  - Evaluate lighting (emission) to 3D texture
  - (High dynamic range is important)
  - Blur texture in 3 dimensions
    - Separable blur
  - Read blurred texture during ray march
    - Mix with original
- Similar to Crytek light propagation volumes



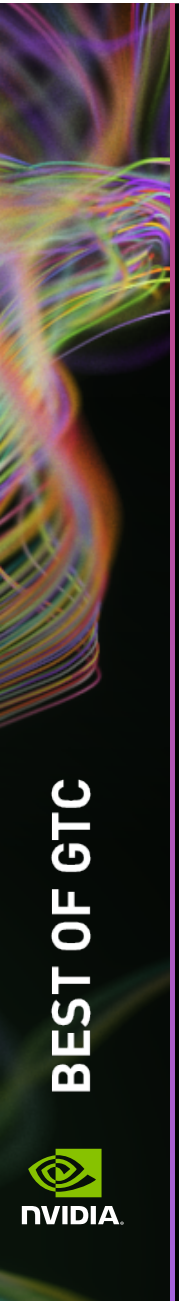
REFLECTIONS

- Easy to cast arbitrary rays though volumes
- E.g. Reflections of fire in other objects
- Calculate reflection ray, intersect with volume bounding box, then ray march through volume
- Can use a down-sampled volume (mipmaps) for blurry reflections

RAY MARCHED REFLECTIONS

BEST OF GTC

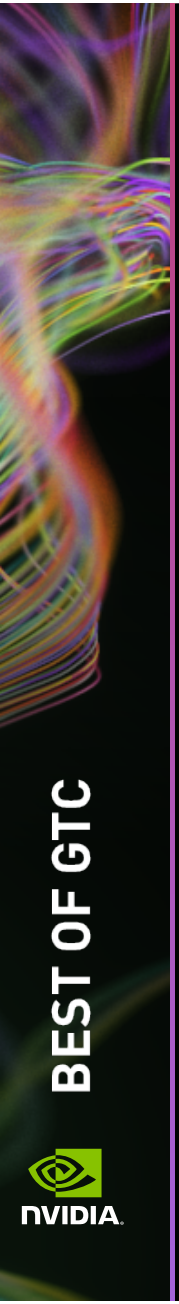
NVIDIA



SIMULATION PERFORMANCE

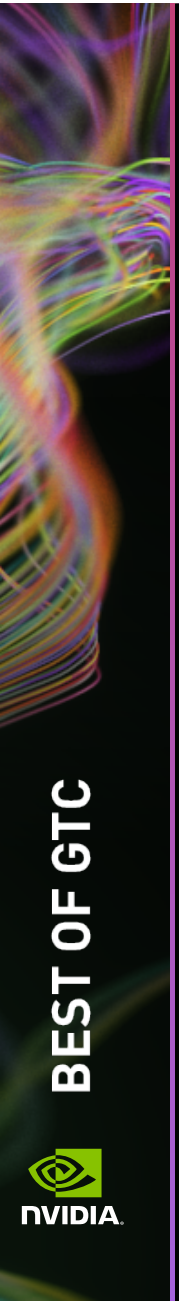
X	Y	Z	Voxels	Grid size (4 x fp16)	Time (msecs)
64	64	64	256K	2 MB	1.0
128	64	64	512K	4 MB	1.7
128	128	128	2M	8 MB	5.1
256	128	128	4M	32 MB	10.7
256	256	256	16M	128 MB	40.0

GeForce GTX Titan
Numbers subject to change.



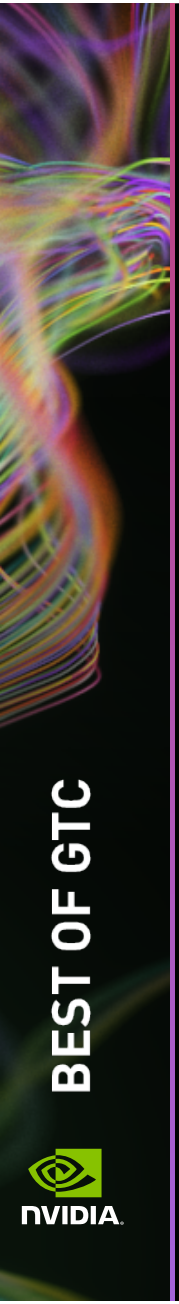
NEW SINCE GTC!

- Optimization
- Forward advection
  - Good for divergent velocity fields
  - Magical effects etc.
- CPU readback
  - Allows use for off-line rendering
  - Caching of simulations



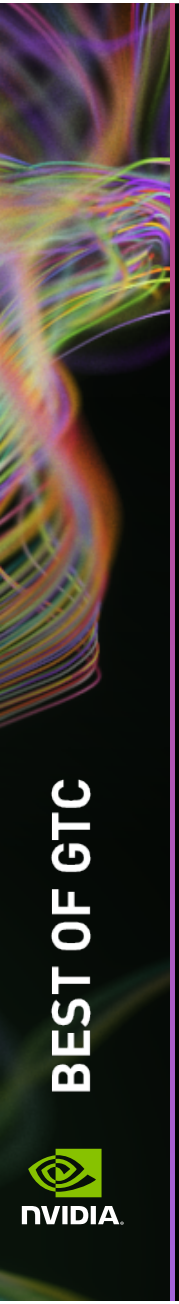
FUTURE WORK

- More optimization
- Improved volume rendering
  - More general lighting
  - Frustum buffers for compositing multiple volumes
- Block sparse volumes
- Global illumination (integration with GI-Works)
  - Fire acts as volumetric light source
- Z-buffer collisions
- Simulation in the cloud?



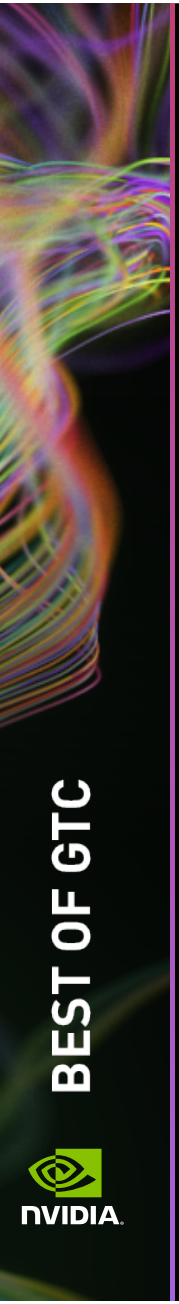
LIVE DEMO

- REAL-TIME LIVE!
  - Tuesday 5:30pm West, Ballroom C-D



QUESTIONS?

- Twitter: @simesgreen



REFERENCES

- Stable Fluids by Stam J. 1999
- Visual Simulation of Smoke by Fedkiw R., Stam J.and Jensen W. H. 2001
- Animating Suspended Particle Explosions by Feldman B. and O'Brien J. 2003
- Advected Textures by Fabrice N. 2003
- Fast Third-Order Texture Filtering by Sigg C. and Hadwiger M. 2005
- Low Viscosity Flow Simulations for Animation by Molemaker J., Cohen M. J., Patel S. and Noh J. 2008
- An Unconditionally Stable MacCormack Method by Selle A., Fedkiw R., Kim B., Liu Y. and Rossignac J. 2008
- Water Flow in Portal 2 by Vlachos A. 2010
- Flame On: Real-Time Fire Simulation for Video Games by Green S. 2012