Mastering DirectX 11 with Unity

Renaldas Zioma
Simon Green

Thursday, March 15, 2012

Overview

- Unity DirectX 11 Renderer
  - Renaldas Zioma, Unity Technologies
  - Twitter: @__ReJ__
- New features and DirectX 11 Effects
  - Simon Green, NVIDIA

New Unity Features

- Unity DirectX 11 Renderer
- Unity “Physically Based” shaders
- Modified lighting pipeline
- Catmull-Clark subdivision
- Blend shapes, PointCache (PC2) support
- Reflection (cubemaps, quads) probes
- Post processing
- Incremental lightmap baking

Thursday, March 15, 2012

Hello, this is a short list of main features that we have developed recently. I’ll mostly talk about Physically Based shaders and Catmull-Clark subdivision.

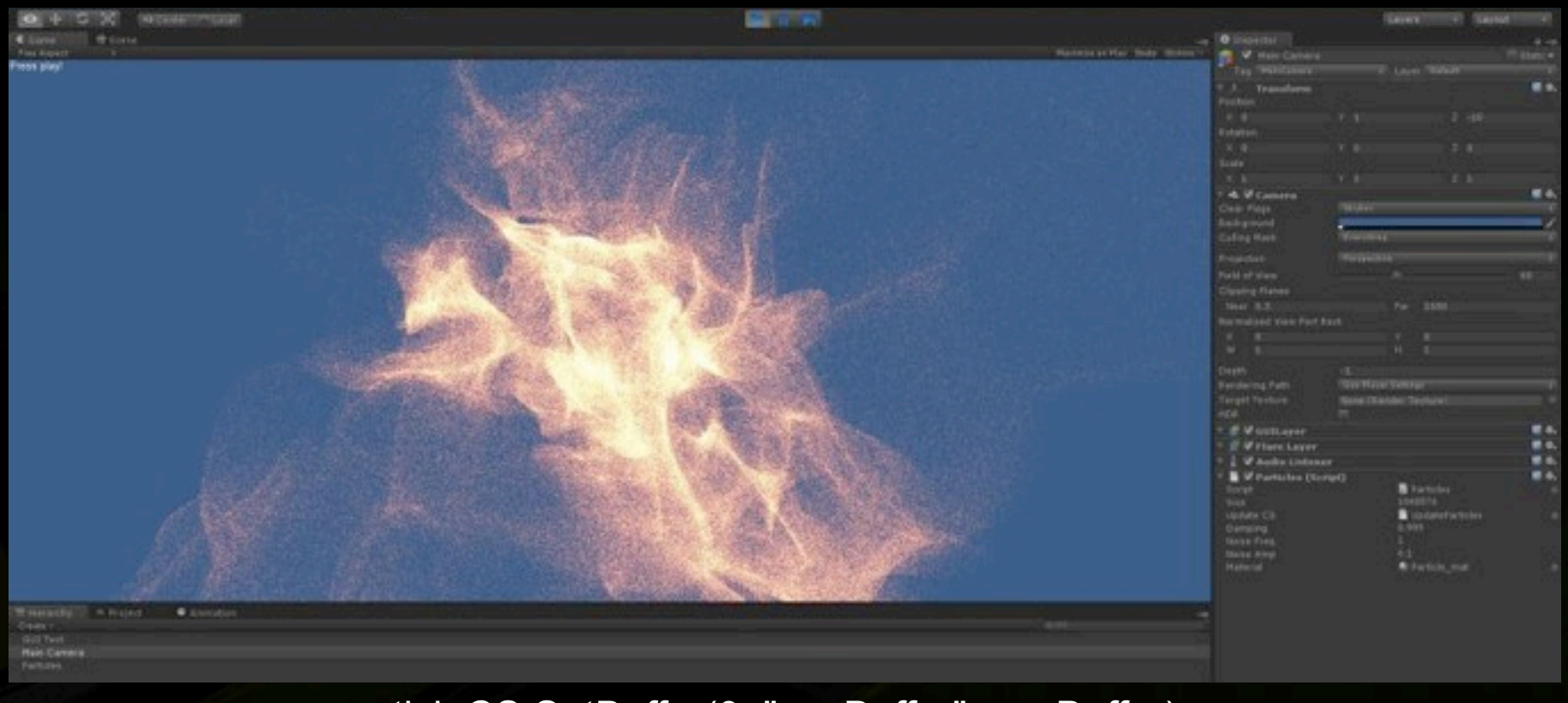
Unity DirectX 11

- Compute shader
- Tessellation
- 3D textures
  - other technologies from 2002 that we added: single channel float textures!

Thursday, March 15, 2012

We have implemented DirectX11 renderer in Unity. Main highlights are compute shaders and tesselation. We finally bothered to implement 3D texture support too.

Compute example - 1M Particles in Unity

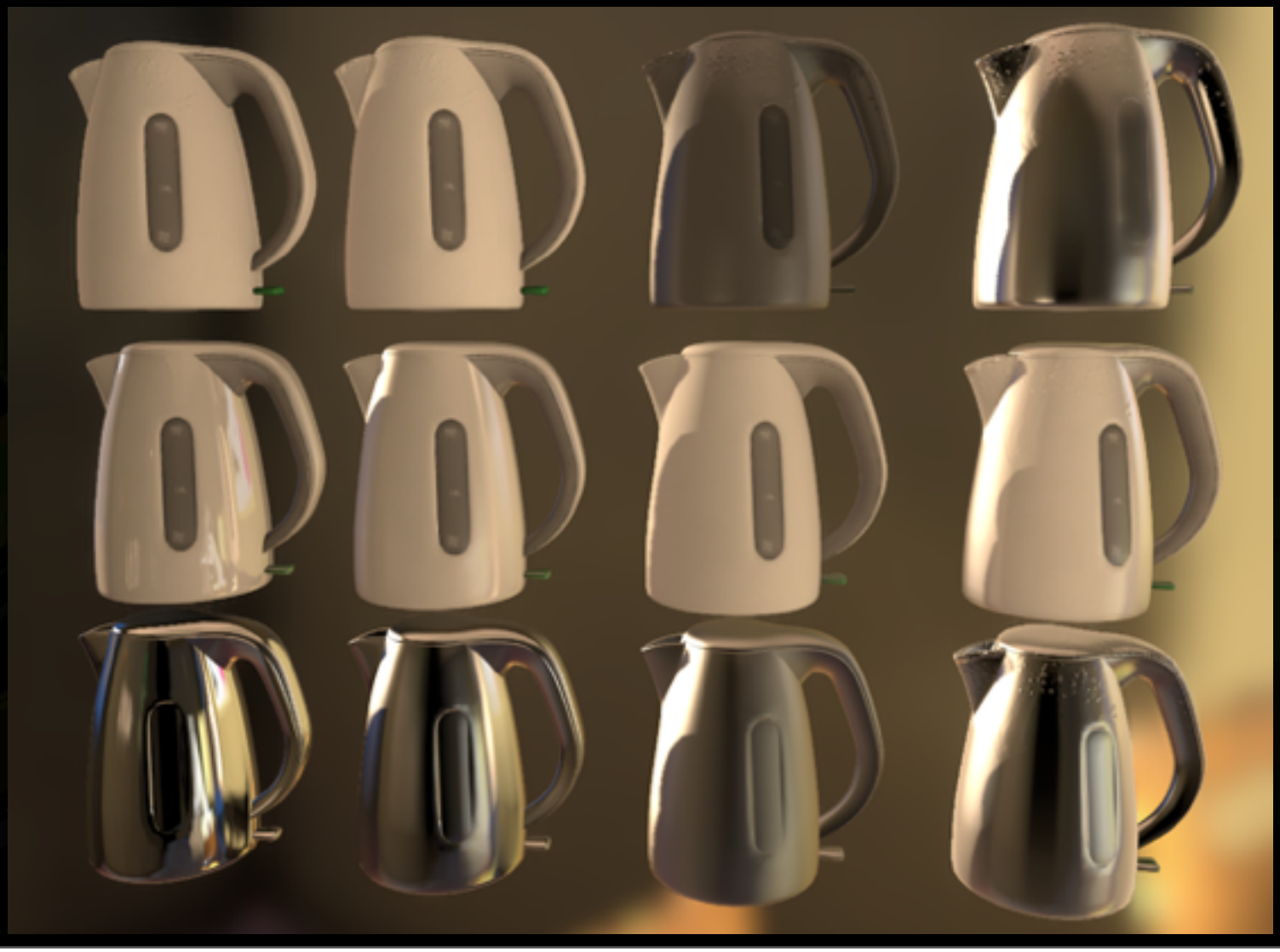


```
particleCS.SetBuffer(0, "posBuffer", posBuffer);
particleCS.SetBuffer(0, "velBuffer", velBuffer);
particleCS.SetFloat("dt", Time.deltaTime);
...
particleCS.Dispatch(0, size, 1, 1);
```

Thursday, March 15, 2012

Compute shader API is really simple and can be accessed directly from C#/Javascript.

Unity “Physically Based” shader



Unity “Physically Based” shader

- Inspired by Mental Ray “Architectural” (MIA) shaders
  - good enough for high quality offline rendering
  - most hard-surface materials (metal, wood, glass, clay)
- Familiar to CG / non-game artists
- Physically based
  - reduces number of parameters
- Energy conserving
  - (kinda) impossible to setup material breaking laws of physics
- Predictable results
  - avoid “hacks”
  - intuitive
- Just 1 shader for almost everything

Thursday, March 15, 2012

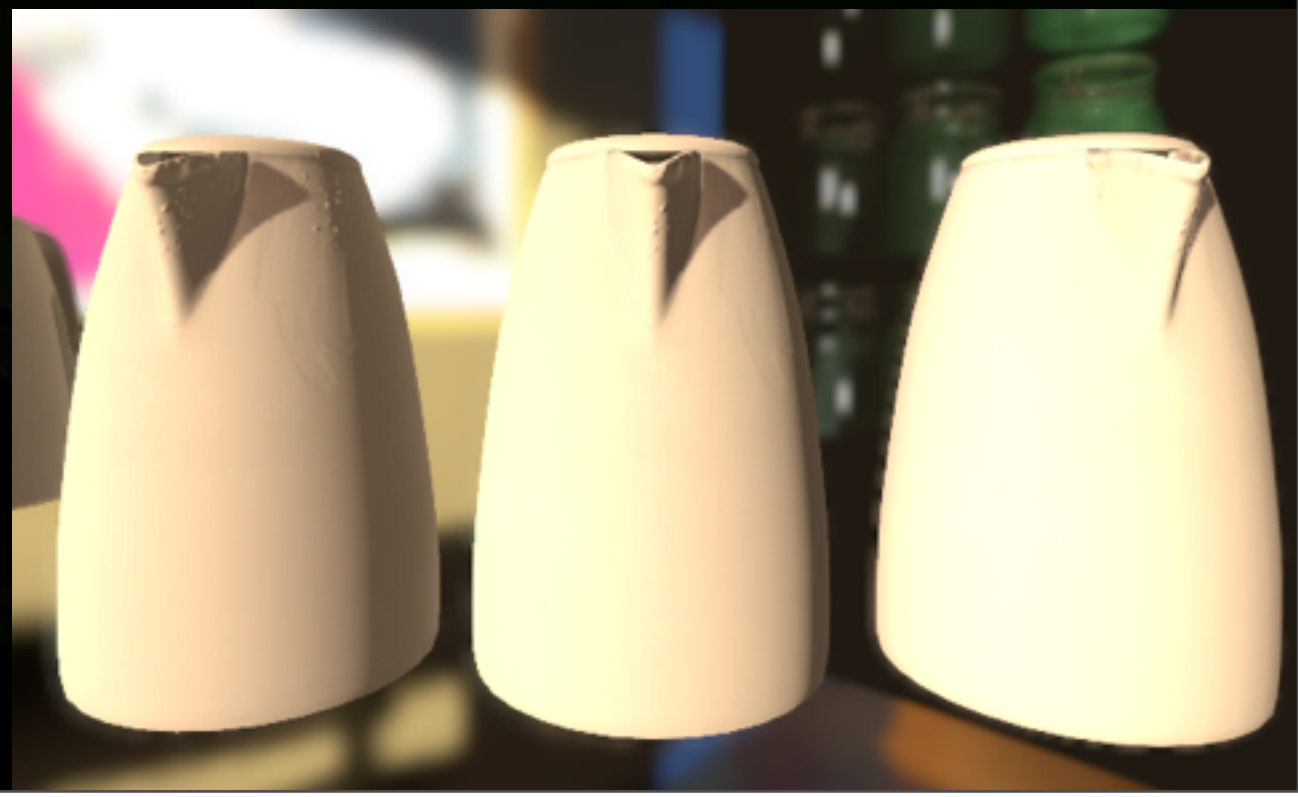
It is important to get shading and lighting foundation right. We can’t really fix broken image with shiny posts.
Modern hardware provides abundant power (esp ALU, texfetch) which can be used for physically based shading. I believe that by avoiding ad-hoc solutions in shading we can achieve more intuitive and predictable workflow for artists.

As an inspiration we have picked MR Architectural shader library - mainly because of familiarity to CG artists, wide range of supported materials while keeping code kernel small and simple and energy conserving.

Our shaders primarily attempts to be physically accurate by keeping the track of the light distribution - light is either absorbed, reflected or transmitted further, hence no light magically (dis)appears.

Unity “Physically Based” shader

- Diffuse: Oren-Nayar
  - 1 parameter: roughness
  - Lambert (roughness == 0)
  - different approximations



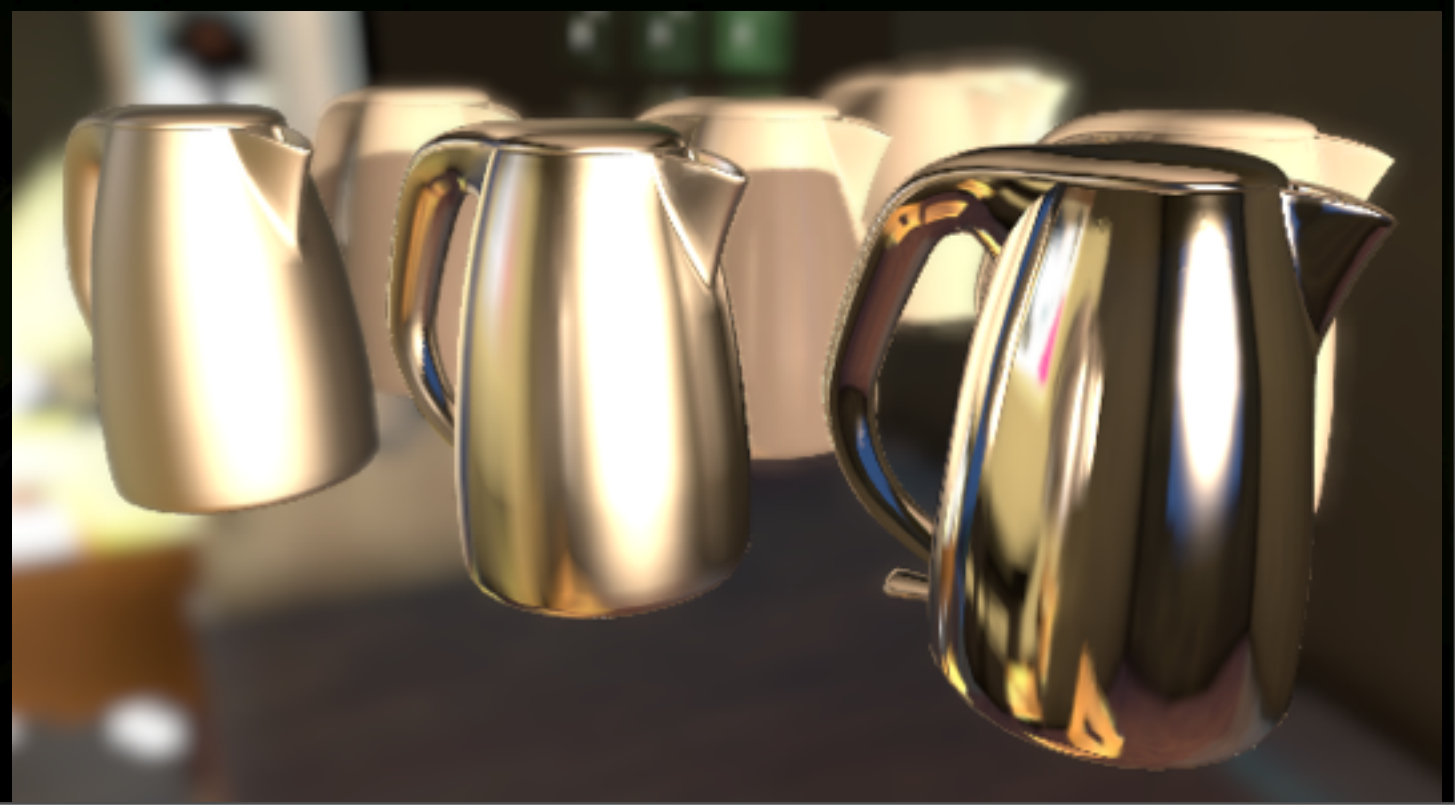
Thursday, March 15, 2012

Oren-Nayar allows rough and powdery surfaces - brick, concrete, clay. Moon surface is another good example of non-Lambertian scattering. OrenNayar can also be used for clothing materials such as velvet. And when roughness is 0, Oren-Nayar degenerates into Lambert.

There are several Oren-Nayar approximations varying in cost and quality.

Unity “Physically Based” shader

- Specular: Cook-Torrance
  - 2 parameters: roughness (glosiness), reflectivity
  - energy conserving
  - physically based
    - micro-facet theory

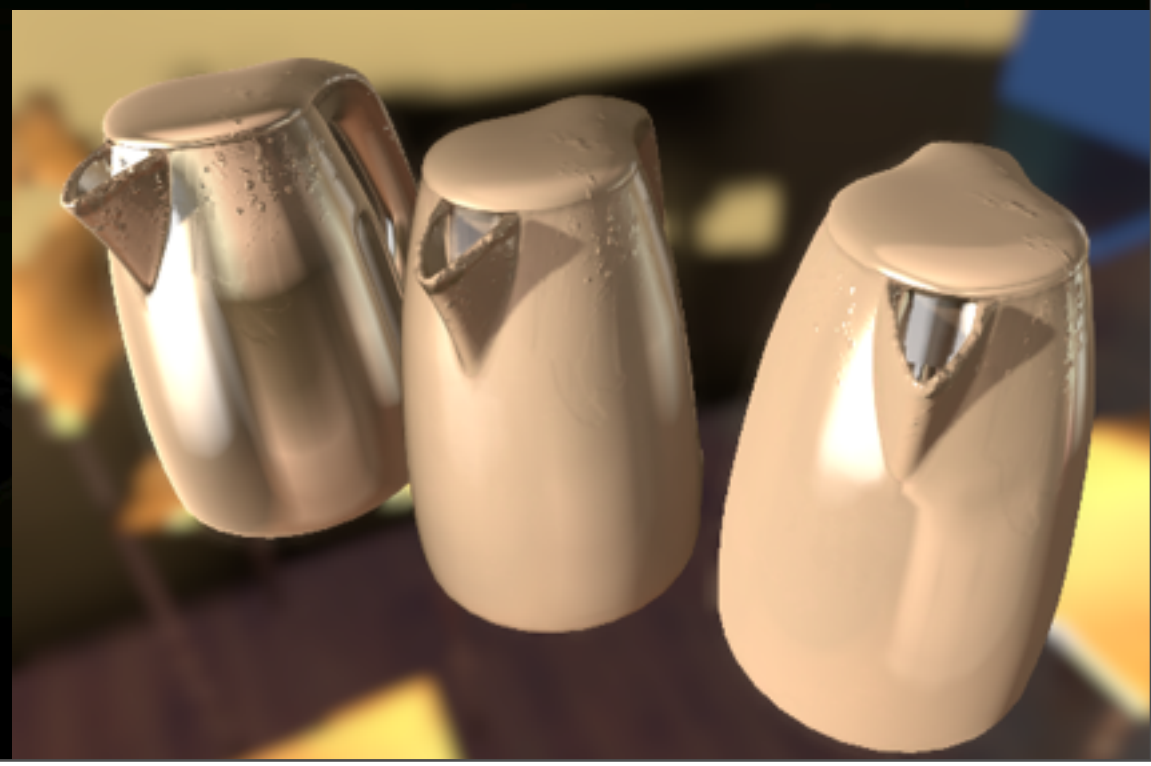


Thursday, March 15, 2012

Cook-Torrance lighting is targeted at metals and plastics and allows to represent specular reflectivity for rough surfaces. Unlike Phong, Cook-Torrance proves accurate way of computing the color of the specular term – an approach based more on physics than pure mathematics.

Unity “Physically Based” shaders

- Fresnel curve - how reflectivity depends on view angle
  - 2 parameters: facing camera (at 0 degrees), at 90 degrees to camera
  - interpolated using Schlick approximation: lerp ($Refl_{at0}$, $Refl_{at90}$, $(N \cdot V)^5$)
  - called “BRDF” in MIA



Thursday, March 15, 2012

View dependent reflectivity - “Fresnel curve” is a simple way to simulate full Bi-directional Reflectance Distribution Function.

Energy conservation

- Diffuse + Reflection (+ Refraction) <= 1
  - 1st law of thermodynamics
- Reflectivity takes energy from both Diffuse and Transparency
  - adding reflectivity will reduce diffuse energy
  - 100% reflective will never be diffuse or transparent!
- Transparency takes energy from Diffuse
  - standard Alpha blending
- Both Cook-Torrance and Oren-Nayar are energy conserving
  - unlike original Blinn-Phong
  - intense highlights are narrow, wide highlights are less intense

Thursday, March 15, 2012

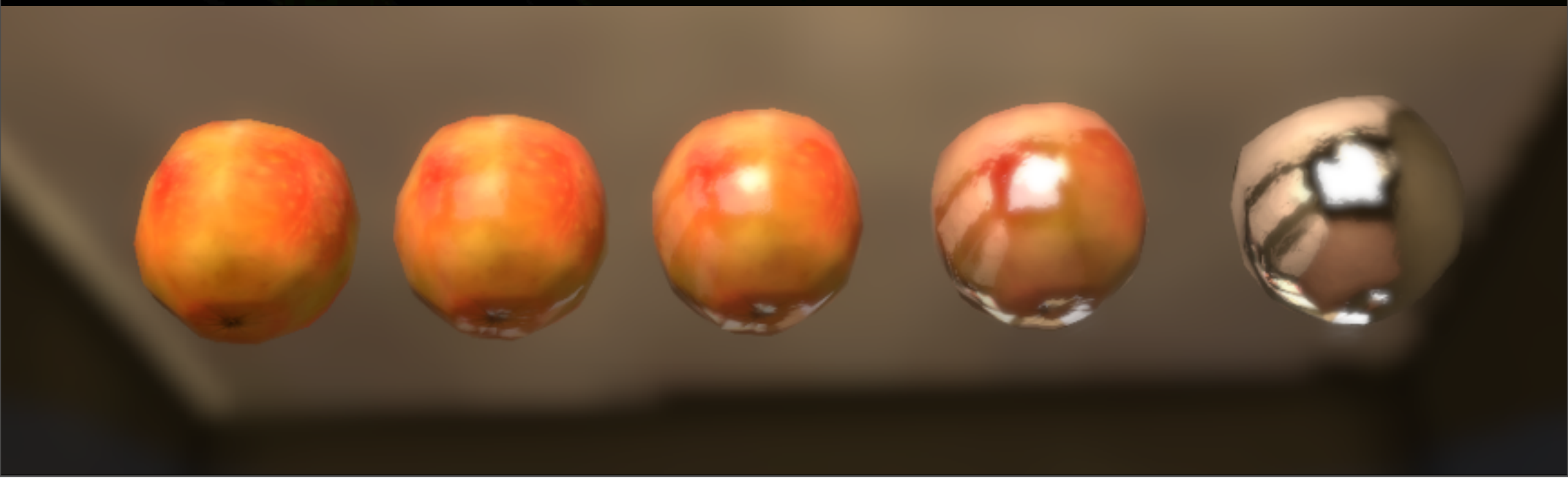
Energy conservation - one of the most important features of the material.
No additional energy is created and the incoming light energy is properly distributed to the diffuse, reflection and refraction components in a way that maintains the 1st law of thermodynamics.

Adding more reflectivity (as energy must be taken from somewhere) the diffuse level and the transparency will be reduced automatically. Adding transparency will reduce diffuse level.

Energy conservation applies to specular highlight of CookTorrance making intense highlights narrow and wide highlights less intense. Analogous behavior is achieved with modified energy conserving BlinnPhong.

Reflectivity takes energy from Diffuse and Transparency

- Adding reflectivity will reduce diffuse energy



Thursday, March 15, 2012

Apple Terminator

Reflectivity varies from 0 on the left to 1 on the right

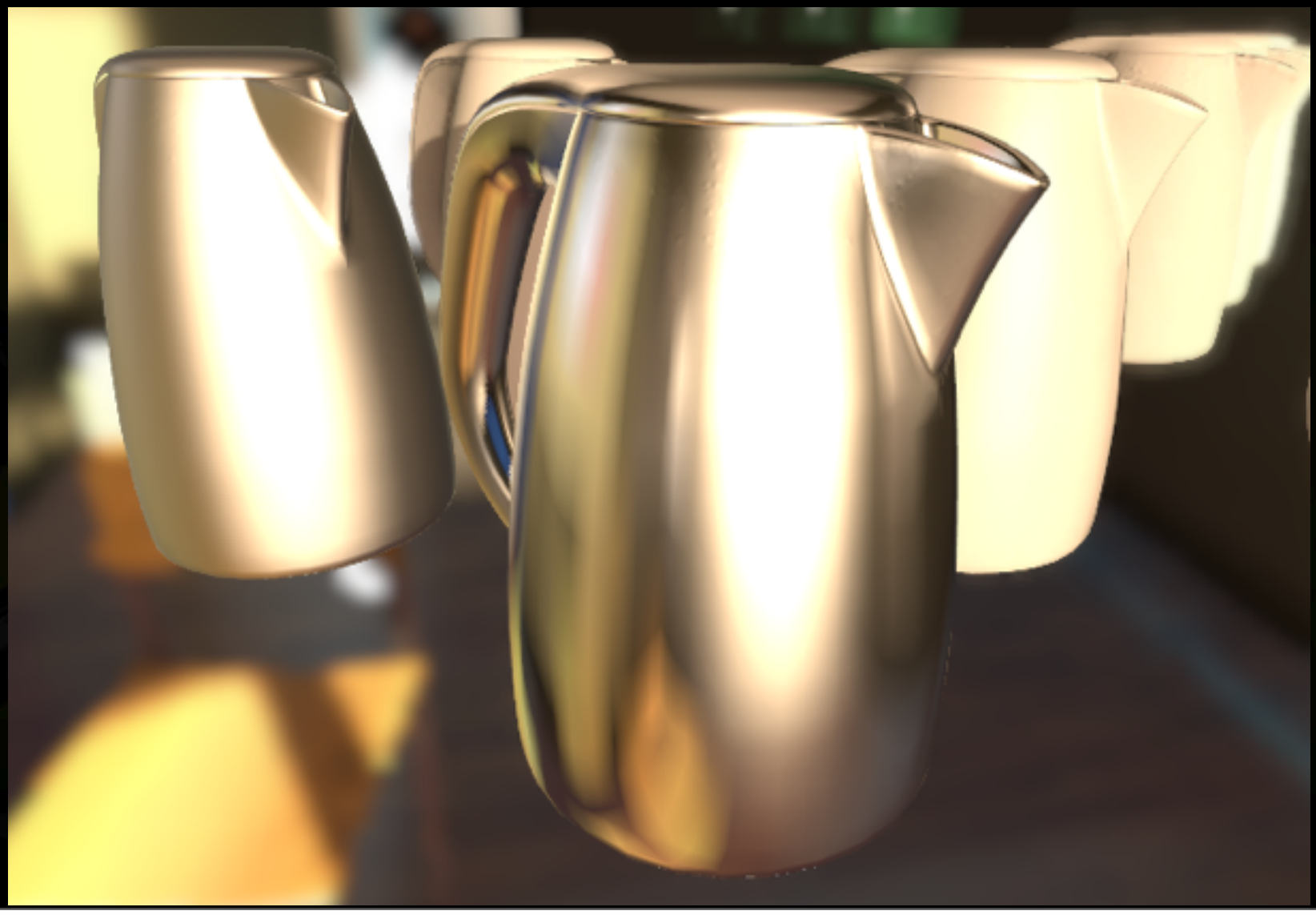
Reflectivity takes energy from Diffuse and Transparency

- 100% reflective will never be diffuse or transparent!

Thursday, March 15, 2012

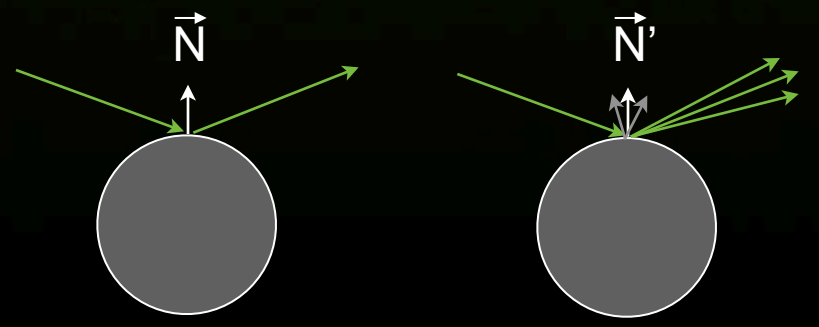
Reflectivity varies from 0.1 on the top to 0.7 on the bottom
Transparency varies from 0.1 on the left to 0.8 on the right

Blurry reflections



Blurry reflections

- Low quality blur by sampling different miplevels (LODbias)
  - DX9/GL requires fixup across cubemap edges
  - does not work for planar reflections
  - only Box filtering
- Increase blur quality by sampling multiple times (1..8)
  - Bend surface normal
    - emulates micro-facets of the rough surface
  - Reflect view direction against “bent” normal
  - Can simulate various normal distributions



Thursday, March 15, 2012

DirectX11 saves headache of fixing up pixels when sampling on the edge of the cubemap.
Using lower miplevels doesn’t work well for planar reflections as they are usually rendered in screen space. Blurring should offset rays along the reflection vector, but not in screen space.

Multiple jittered samples could be used to improve quality of the blurry reflections beyond box filtering.

Combining 2 normal maps

- Artists wanted detail normal maps
- Blending 2 normal maps just “flattens” both
- Want to get results as if blending 2 heightmaps
- “Warp” 2nd normal map
  - using normals from the 1st normal map:

```
float3x3 nBasis = float3x3(
        float3 (n1.z, n1.x,-n1.y),
        float3 (n1.x, n1.z,-n1.y),
        float3 (n1.x, n1.y, n1.z ));
n = normalize (n2.x*nBasis[0] + n2.y*nBasis[1] + n2.z*nBasis[2]);
```

Without normals

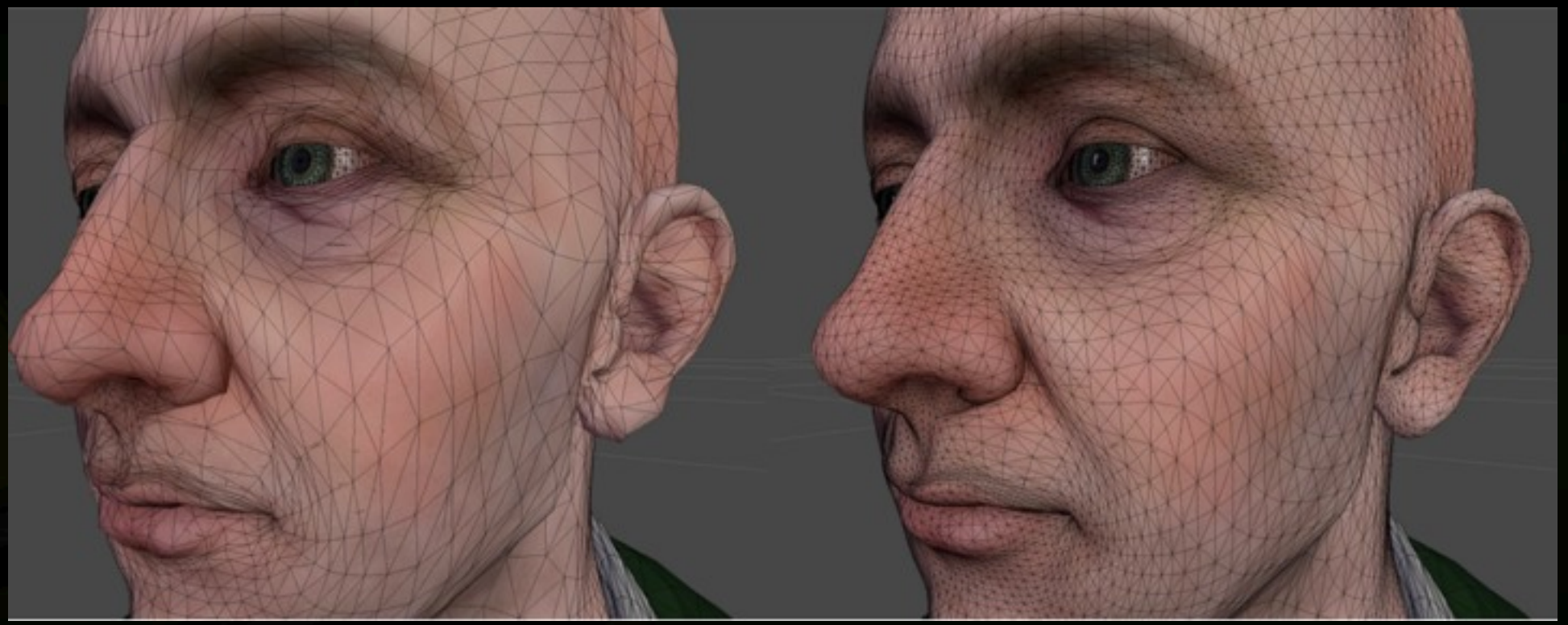
Detail normals

Combined normals

Optimizing “Physically Based” shaders

- Small code kernel - easy to optimize
  - same code for Skin and Car shading too
- Costs
  - 90 ALU instructions on average
  - 270 ALU instructions at max
  - up to 24 texture fetches
- Permutations - pick depending on parameter values
  - if diffuse roughness = 0, use Lambert instead of OrenNayar
  - different OrenNayar approximations
  - glossiness value affects number of reflection texture samples
  - etc

Catmull-Clark Subdivision



Catmull-Clark Subdivision

- Standard in modeling and offline rendering
- Does NOT map directly to DirectX11 tessellation pipeline
- PN-Triangles or Phong Tesselation does not exactly match results from modeling tools
- Ongoing research on Catmull-Clark approximation on GPU
  - requires complex mesh preprocessing
  - or too expensive for required quality

Catmull-Clark Subdivision

- Instead we perform 1st subdivision level on CPU
  - most visually important subd level
- 7.4M verts/sec
  - per single core on Core i7 2600K
  - plain C implementation
  - not fast, but works!
- Higher subd levels on GPU using Phong Tessellation
- Future plan: move Catmull-Clark code to COMPUTE shader
  - otherwise bandwidth bound due to CPU -> GPU uploads

Thursday, March 15, 2012

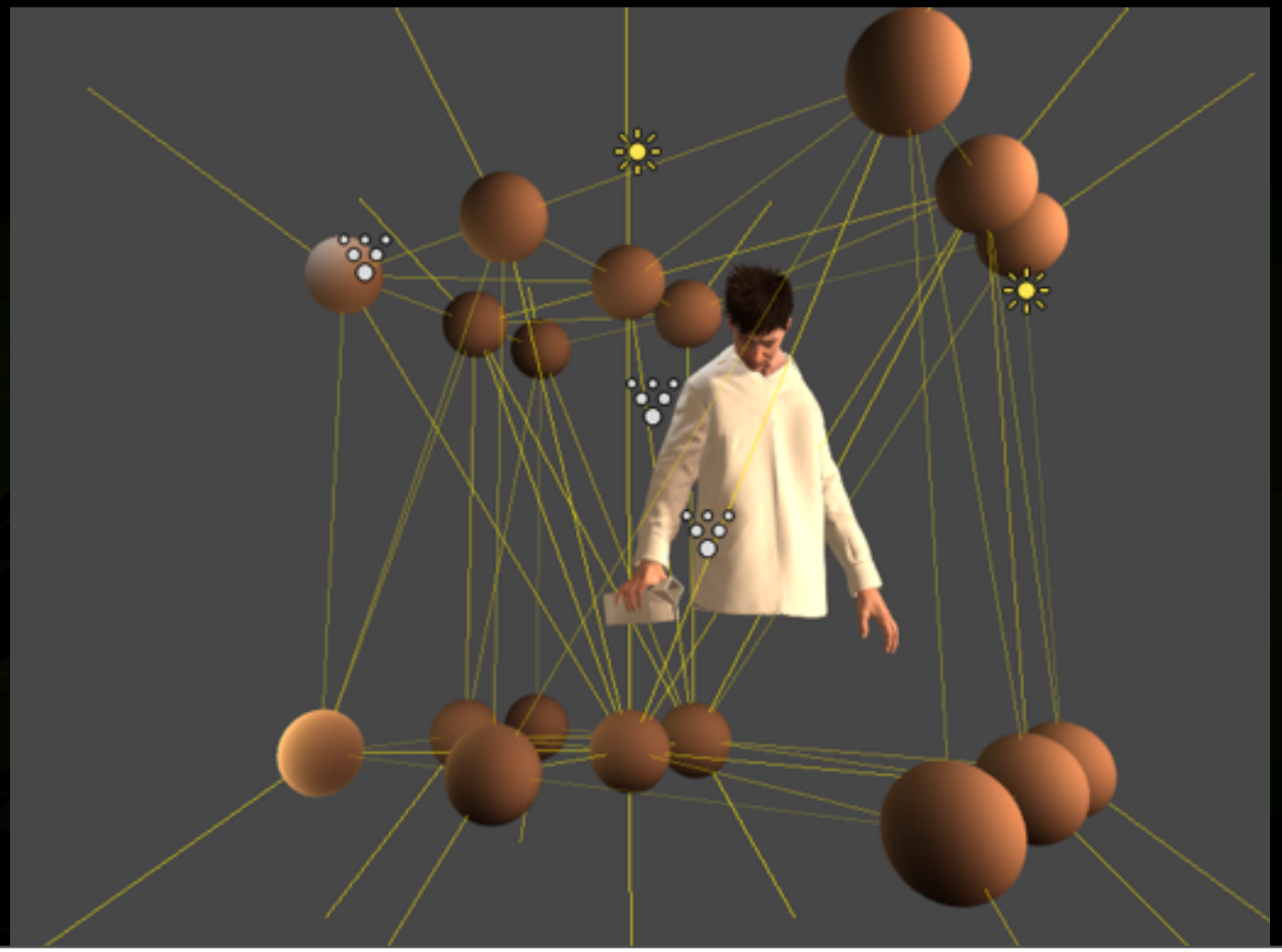
We went for a pragmatic approach: all characters would be modelled with subdivision surfaces like usual in offline CG. Then at runtime, we perform one level of Catmull-Clark subdivision - this is enough to get the surface be approximately the right shape. Subdivision is performed after animation, mesh skinning and blend shapes (morph targets) are calculated, and it computes new vertex positions, normals and tangent vectors.

In our case the initial mesh is already quite dense to support animation, and after one level of Catmull-Clark subdivision the shape is close to the limit surface.

Global Illumination

- Unity supports lightmap baking using Beast
- Only indirect lighting is stored in lightmaps
- Light probes
  - lighting for dynamic objects
- Reflection probes
  - cubemaps (infinite distance reflections)
  - quads (local reflections)

Light probes - tetrahedras



Ambient occlusion

- Horizon Based Ambient Occlusion (HBAO)
  - developed by NVIDIA
- Ambient Occlusion should NOT affect direct illumination
  - by definition!
  - otherwise gets dirty look
- Calculate SSAO early, feed results into pixel shaders
  - min (bakedAO, objectAO, SSAO)
  - apply only to indirect lightmap, light probes and reflections

Thursday, March 15, 2012

Ambient occlusion should affect only indirect (ambient) illumination. Direct illumination should be affected only by direct occlusion (shadowmaps).

Old Unity SSAO

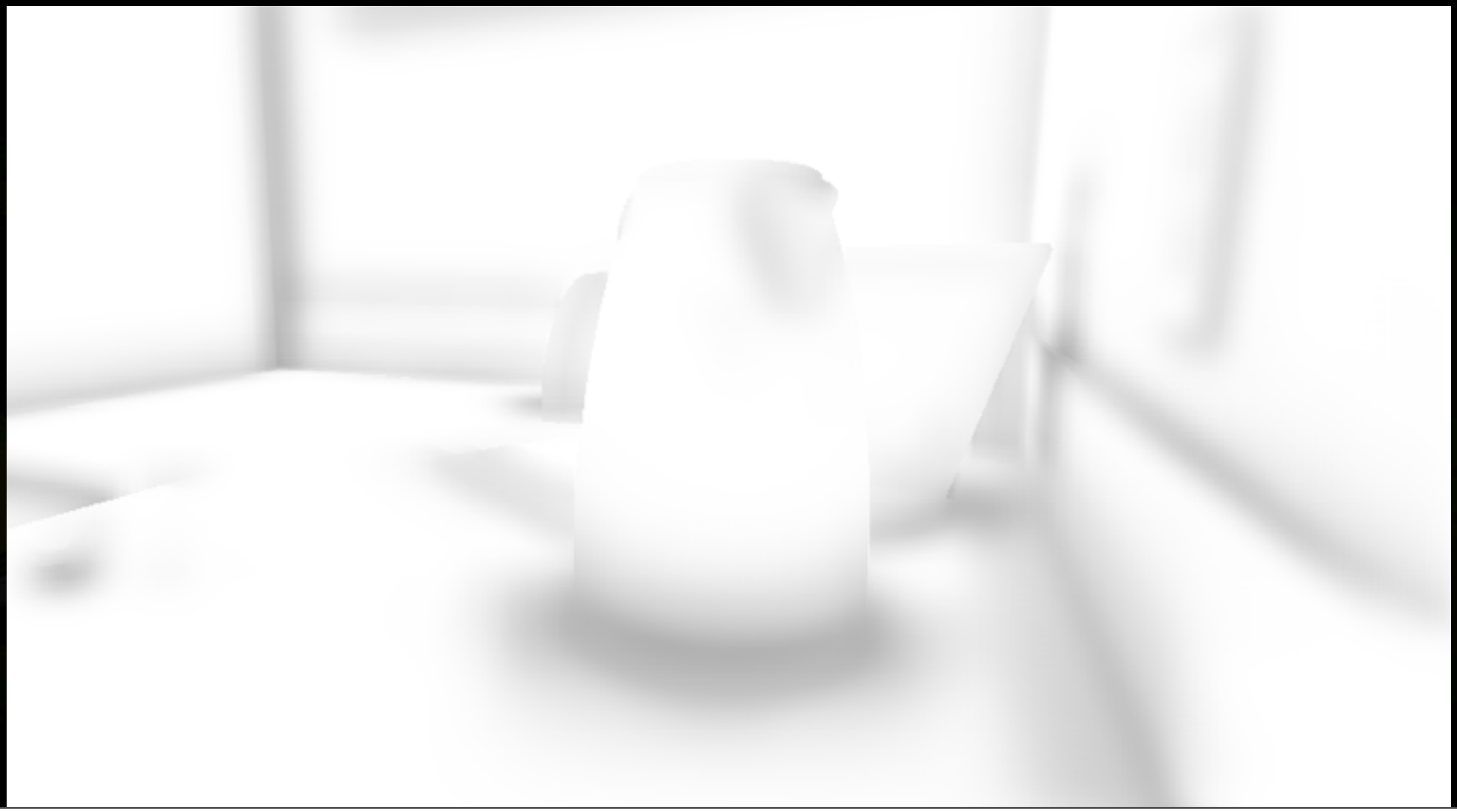
Comparison between old Unity SSAO and new HBAO algorithms

HBAO

Thursday, March 15, 2012

Comparison between old Unity SSAO and new HBAO algorithms

HBAO



Skin rendering

- Based on the same code as hard-surface shader
- But in multiple layers
- “Double” specular
- Texture space diffusion

Mulitple layers with varying blur amounts
One specular layer is used to mask second specular layer – artistic decision

Thursday, March 15, 2012

Thursday, March 15, 2012

eyes could look better ;(

DirectX 11 Effects

- APEX Destruction
- Hair pipeline
- Volumetric explosions
- Motion blur

APEX Destruction

- Unity already uses PhysX
  - Rigid body simulation
- APEX Destruction enables real-time destruction effects
  - Fracturing objects
  - Breaking walls, smashing windows etc.
- NVIDIA integrated APEX runtime into Unity

NVIDIA APEX – Scalable Dynamics Framework

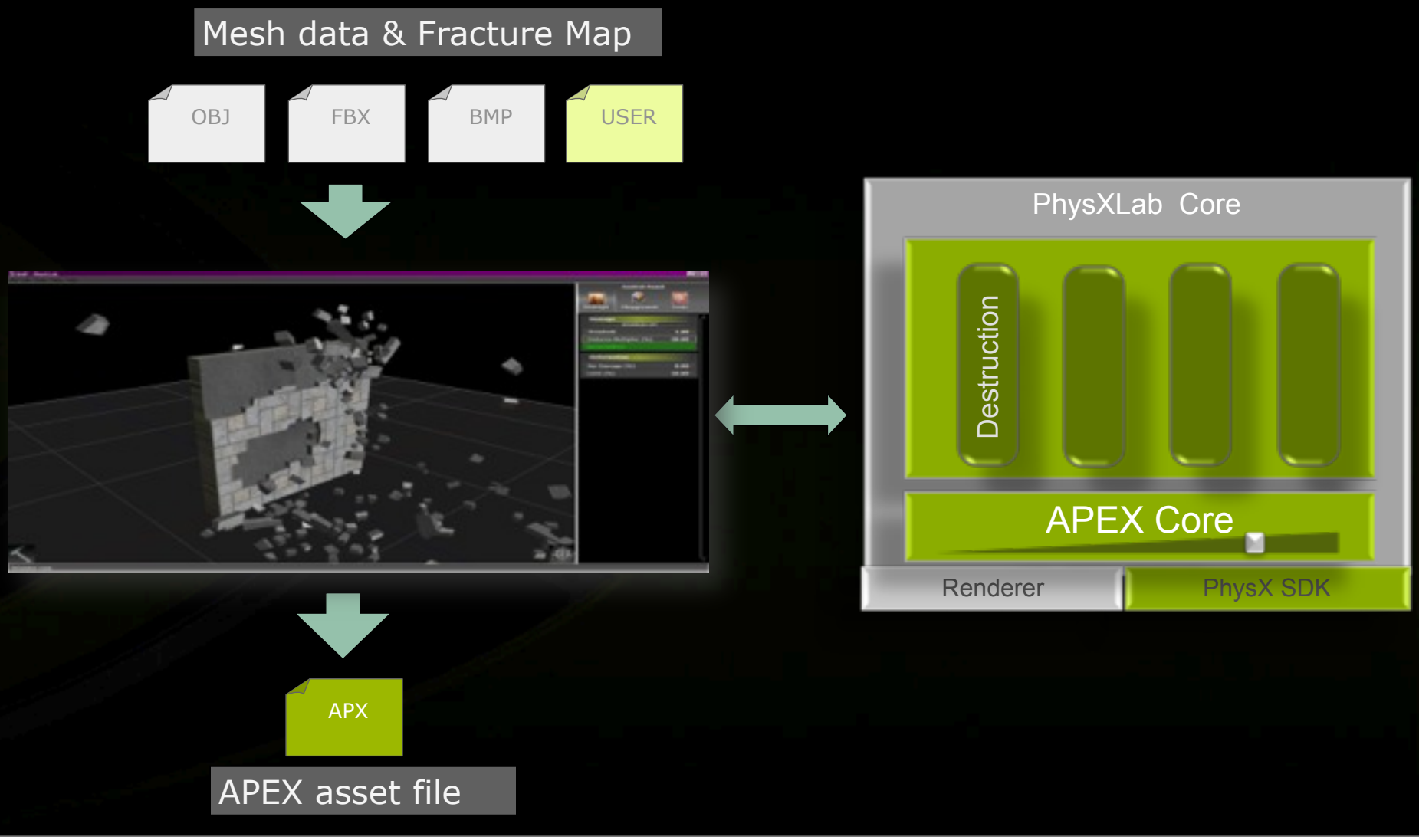
Thursday, March 15, 2012

APEX is a scalable Dynamics Framework (aka Adaptive Physics EXtensions)

Extendible framework: the building blocks are modules that solve certain problems. Their runtime complexity varies, but a common to all is attention to ease of authoring. Apex is artist-focused.

Many modules are currently available, or will be soon. Destruction and Clothing are ready now, in our 0.9 release, as well as particles and forcefields. Vegetation (which integrates with SpeedTree) and Turbulence will be available soon.

APEX Destruction



Thursday, March 15, 2012

Now we'll see how to author destructibles.

Destruction is authored in PhysXLab. PhysXLab is a standalone tool that allows artists to fracture arbitrary objects and immediately try them out within the tool.

You can import for example meshes from Alias OBJ or Autodesk FBX format files, then fracture it using a variety of methods. The result is stored in an Destructible Asset file (pda). This includes the fractured graphics mesh as well as references to resources such as materials and particle systems.

The tool allows you to select the internal material, used for the newly-generated internal faces.

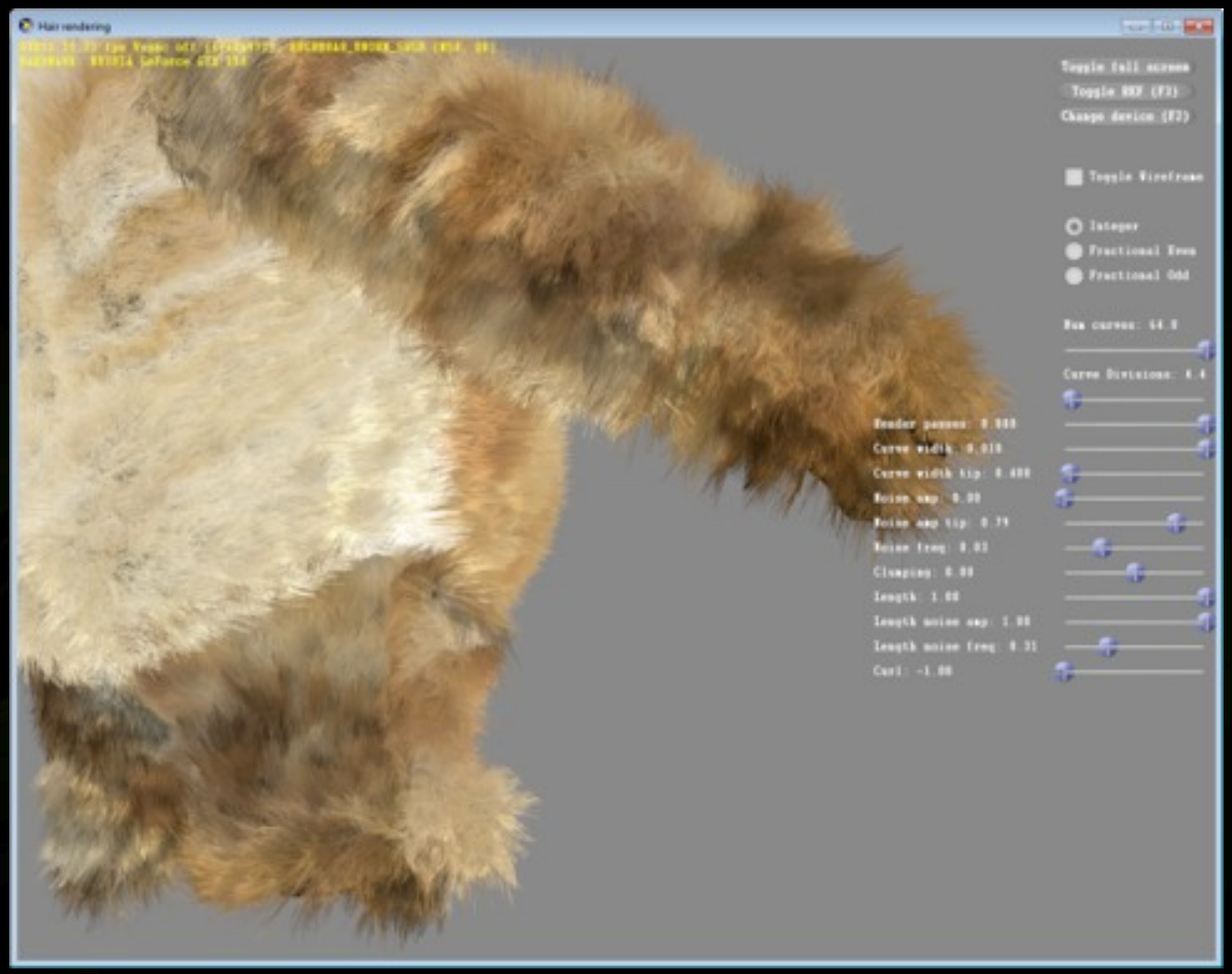
What you see is what you get – the APEX file output is used for simulation preview within the tool, as well as the runtime loadable asset. There's no difference.

APEX Destruction Demo

Hair Pipeline

- Hair and fur is the next big challenge for game characters
  - as animation and skin shading gets better, low quality hair more obvious
- Unity hair system was designed to be similar as possible to off-line systems
  - Hair styles (groom) can be modeled in Softimage XSI / Maya / MAX
  - Exported to Unity using PC2 point cache format
  - Rendered using DirectX 11 tessellation
- Generation of render hair geometry done entirely in hardware
  - Geometry amplification
  - GeForce GTX 580 tessellation hardware is very fast

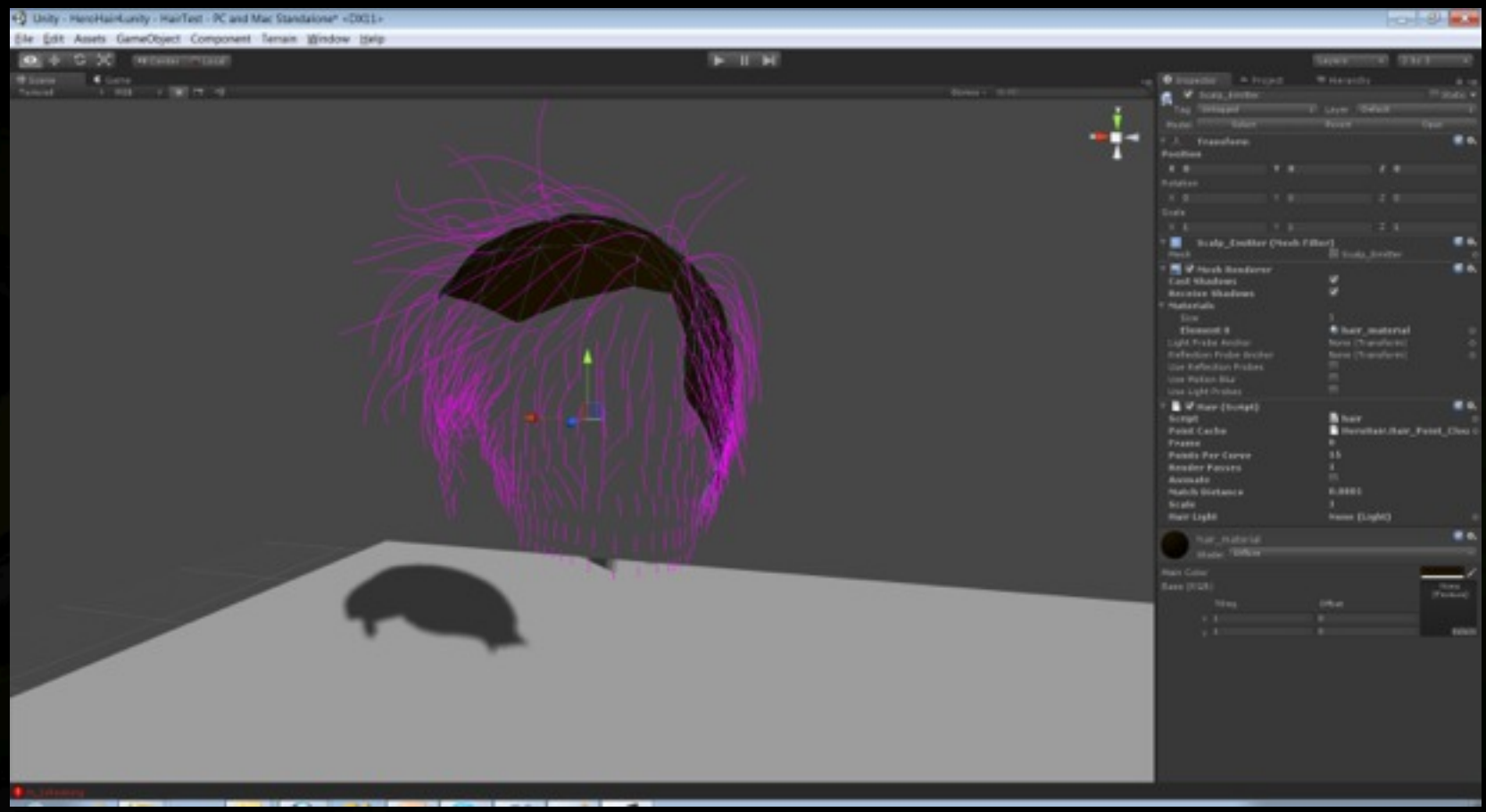
Early DirectX 11 Hair Test



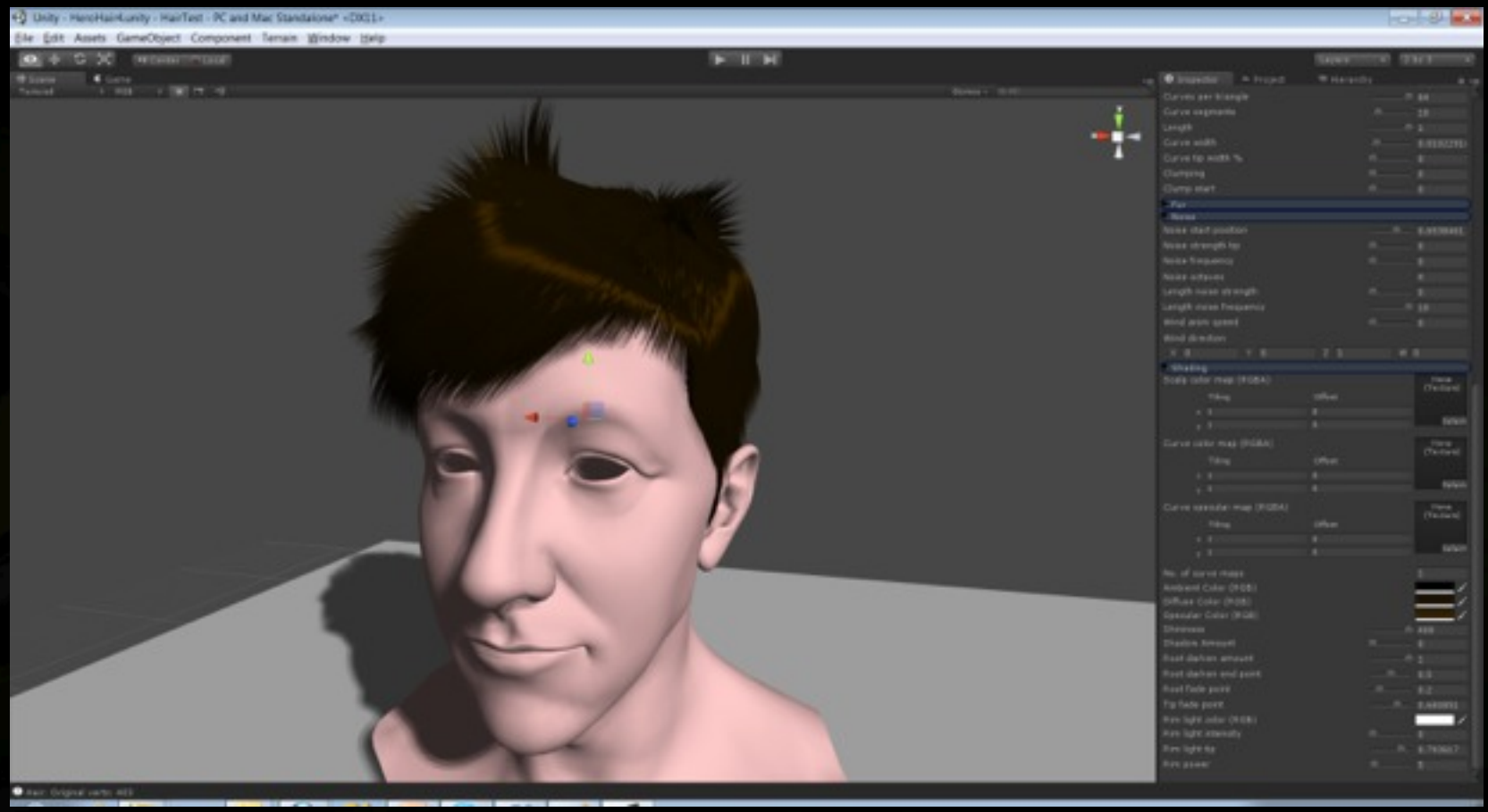
Hair Styling

- Hair defined by two items:
  - Emitter mesh
  - Guide hairs
    - defined at vertices of each triangle in emitter mesh
    - typically 15 control vertices per guide hair
    - stored in DirectX 11 structured buffer

Guide Hair Curves and Emitter Mesh



Generated Render Hairs



Hair using DirectX 11 Tessellation

- Uses “isoline” domain
  - Hardware can generate up to 64 isolines / patch, with up to 64 verts / line
- Domain shader
  - Generates render hairs over surface of each triangle
    - chooses a random position on triangle (using integer hash function)
    - render hairs interpolated between guide hairs using barycentric coords
  - Evaluates position on each curve using B-Spline
  - Number of curves / tri and vertices/curve can be varied
    - supports LOD and back-face culling
  - Use 3D noise based on position to add detail, wind effects
- Geometry shader
  - Expands line strips to camera-facing triangle strips

Using splines allows actual number of vertices generated along curve to be varied

Hair

Thursday, March 15, 2012
after NVIDIA party

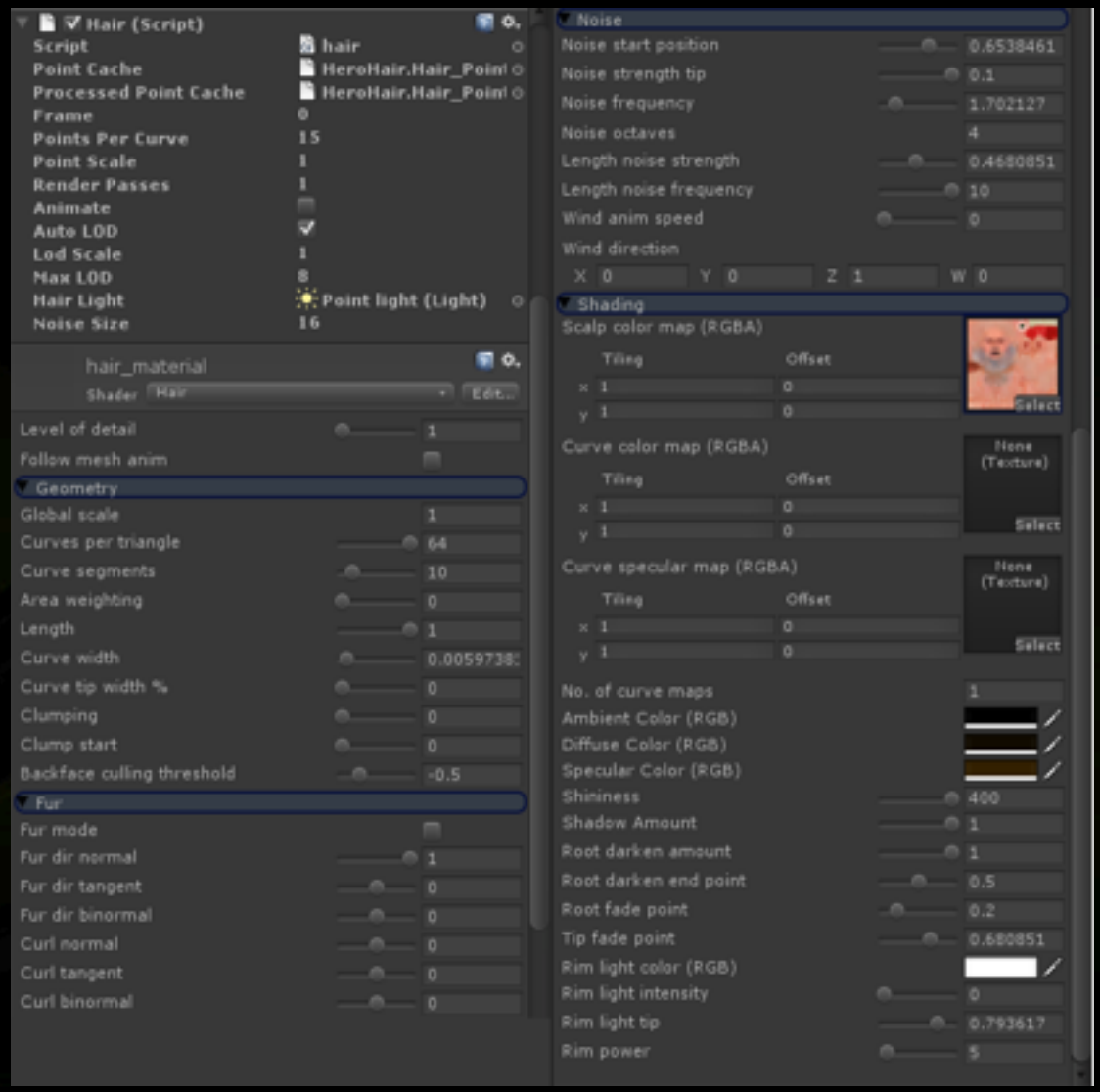
Add Noise

Clumping

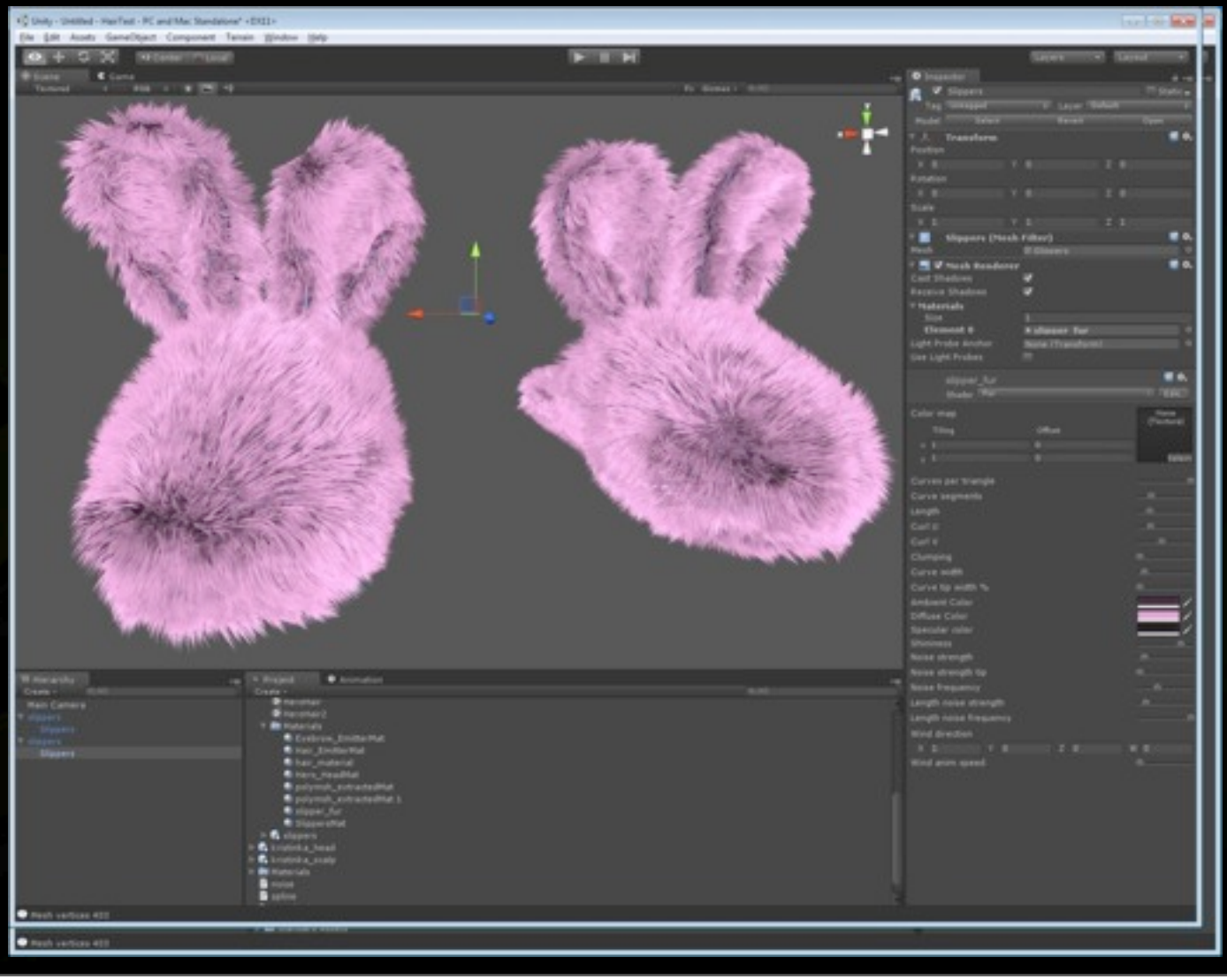
Hair Shading

- System supports thin geometric hair
  - or wide textured strips with images of multiple hairs
- 8x multisample anti-aliasing works surprisingly well
  - use Alpha-To-Coverage with added random dithering
  - gives order independent transparency without blending
  - transparency super-sampling using shader coverage output would also possible, but expensive
  - shader includes fading at root and tip
- Use Kajiya-Kay anisotropic shading model
  - supports self-shadowing
  - fake rim-lighting effect

Hair UI


Hair (Script)
Script
hair
Point Cache
HeroHair.Hair_Point
Processed Point Cache
HeroHair.Hair_Point
Frame
0
Points Per Curve
15
Point Scale
1
Render Passes
1
Animate
Auto LOD
Lod Scale
1
Max LOD
8
Hair Light
Point light (Light)
Noise Size
16
hair_material
Shader
Hair
Edit...
Level of detail
1
Follow mesh anim
Geometry
Global scale
1
Curves per triangle
64
Curve segments
10
Area weighting
0
Length
1
Curve width
0.0059738
Curve tip width %
0
Clumping
0
Clump start
0
Backface culling threshold
-0.5
Fur
Fur mode
Fur dir normal
1
Fur dir tangent
0
Fur dir binormal
0
Curl normal
0
Curl tangent
0
Curl binormal
0
Noise
Noise start position
0.6538461
Noise strength tip
0.1
Noise frequency
1.702127
Noise octaves
4
Length noise strength
0.4680851
Length noise frequency
10
Wind anim speed
0
Wind direction
X 0
Y 0
Z 1
W 0
Shading
Scalp color map (RGBA)
Tiling
Offset
x 1
0
y 1
0
Select
Curve color map (RGBA)
None (Texture)
Curve specular map (RGBA)
No. of curve maps
1
Ambient Color (RGB)
Diffuse Color (RGB)
Specular Color (RGB)
Shininess
400
Shadow Amount
1
Root darken amount
1
Root darken end point
0.5
Root fade point
0.2
Tip fade point
0.680851
Rim light color (RGB)
Rim light intensity
0
Rim light tip
0.793617
Rim power
5

Not just for hair!



Thursday, March 15, 2012

Supports fur mode, doesn't use guide hairs, fur generated based on mesh tangent and normal.

Explosions

- Explosions are a common element in modern video games!

Inspiration – FumeFX

Explosion Approaches

- Particle system
  - Can look good, especially with pre-rendered sprites
  - But hard to get fireball look
  - Sprites can look flat, especially with a moving camera
- Fluid simulation
  - Possible in DirectX 11, but expensive
  - Hard for artists to control
- Procedural techniques
  - Practical on today's PC hardware

Great Presentation from GDC 2005

GameDevelopers Conference

NVIDIA

Conclusion

- Volume rendering is fun
  - Becoming practical for use in games
- Shader model 3.0 looping makes new effects possible

!!

SAN FRANCISCO CA MAR 7-11 GDC 05

Distance Fields

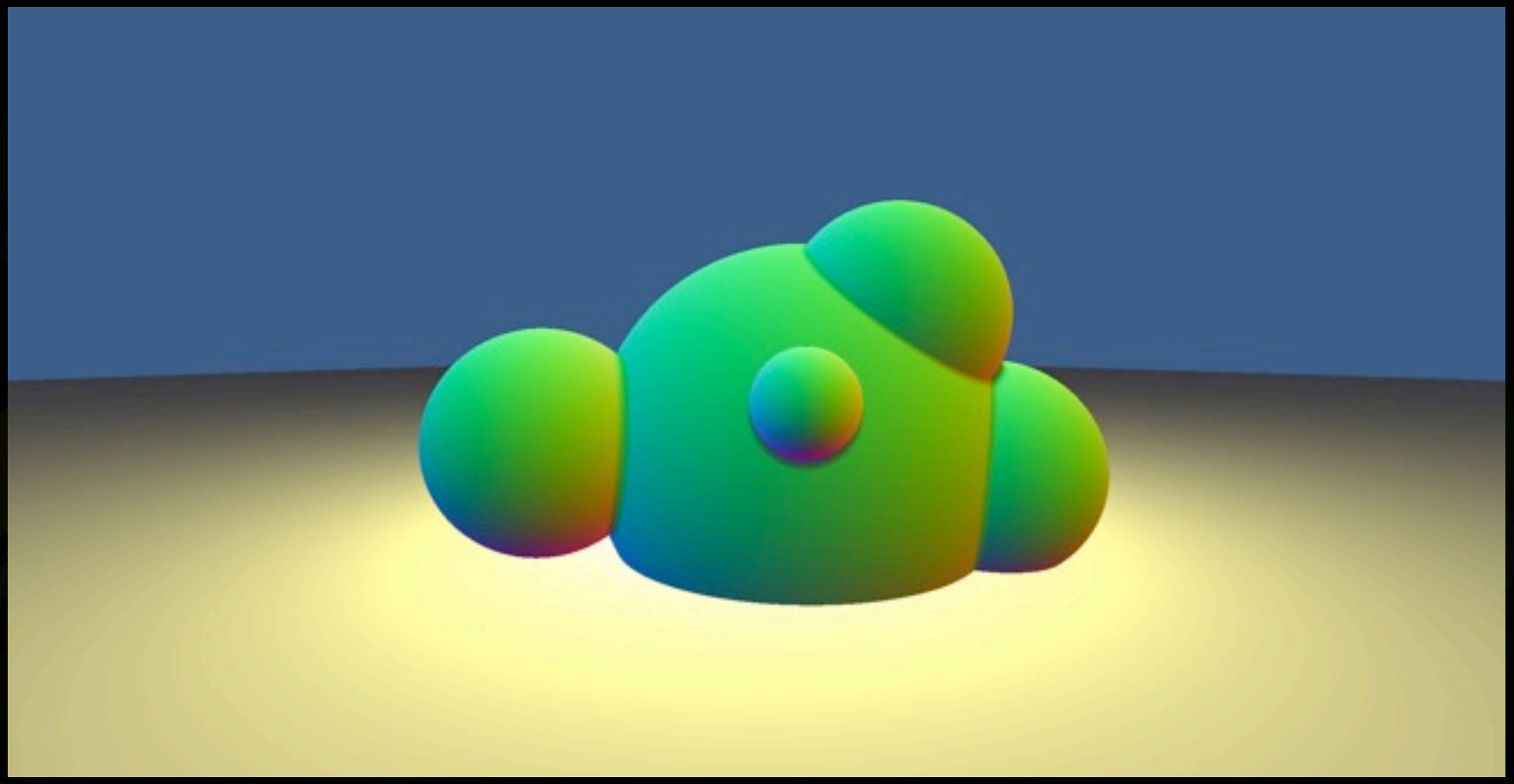
- Aka Level Sets
- Popular in demo scene
- Store distance to nearest surface

- Advantages
  - Can render very quickly using “sphere tracing”
    - ray marching using distance information to take larger steps
  - Can calculate normal as gradient of distance field
  - Nice ambient occlusion approximations

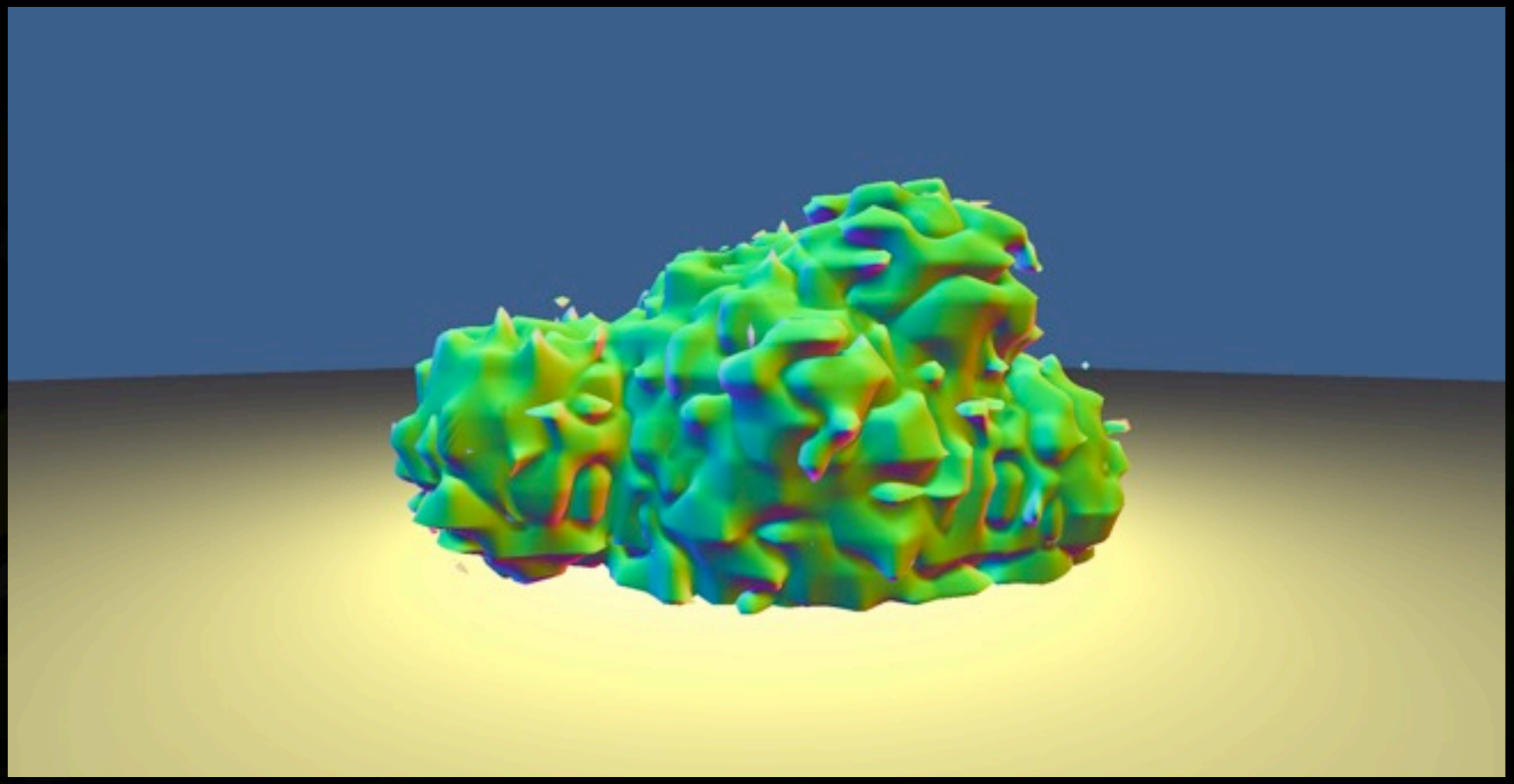
Volumetric Explosions

- Basic shape defined using spheres
  - Spheres can be animated in Unity
- Detail added using “pyroclastic” noise
  - Simply offset distance from surface using Perlin noise at position
  - Noise stored in small 3D texture (64^3). Use 4-6 octaves
- Shade based on displacement amount
- Rendered using hybrid distance field / volume renderer
  - Sphere trace to surface
    - 32 steps often enough
  - Then ray-march through volume
    - ~16 steps

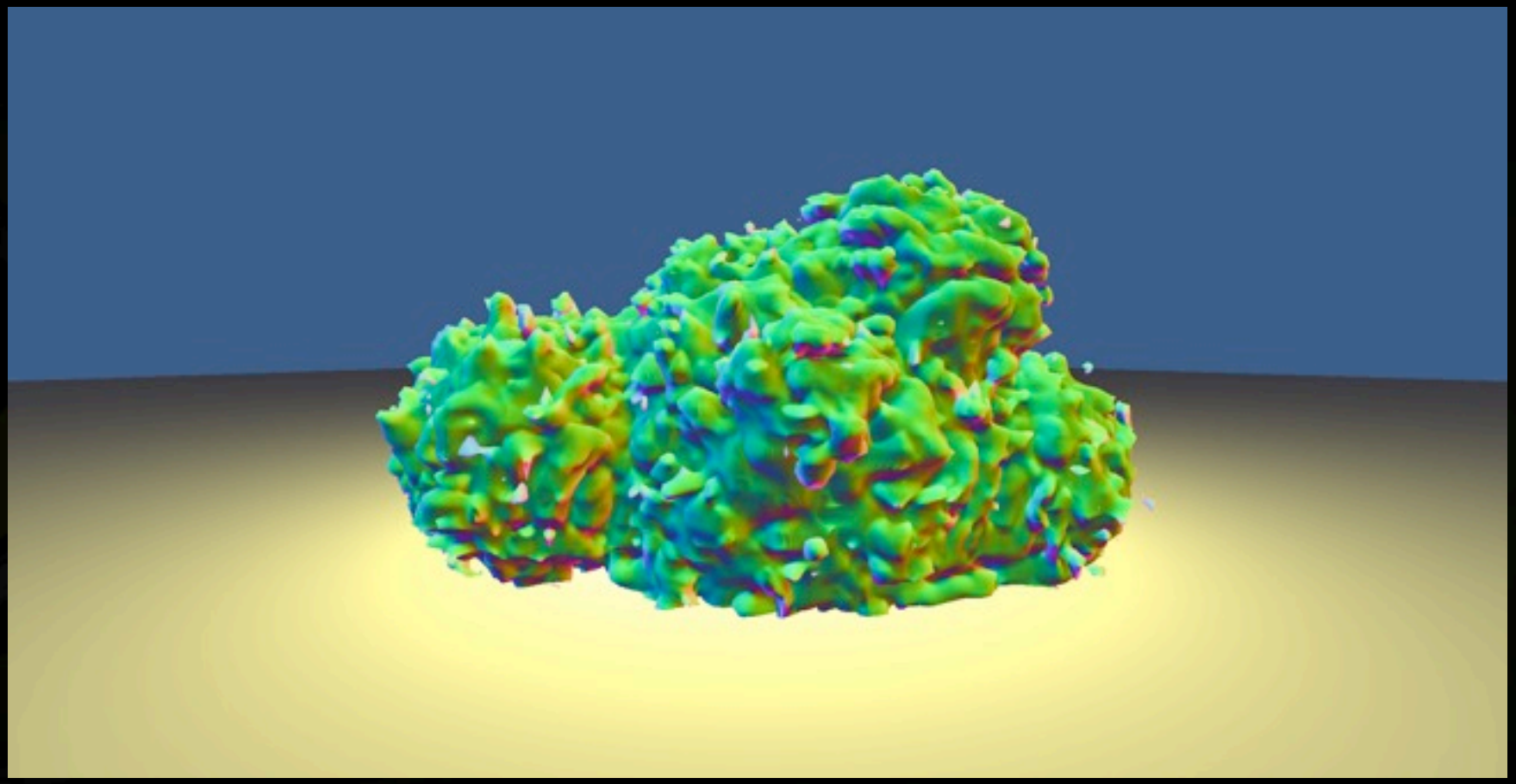
Start with some spheres…



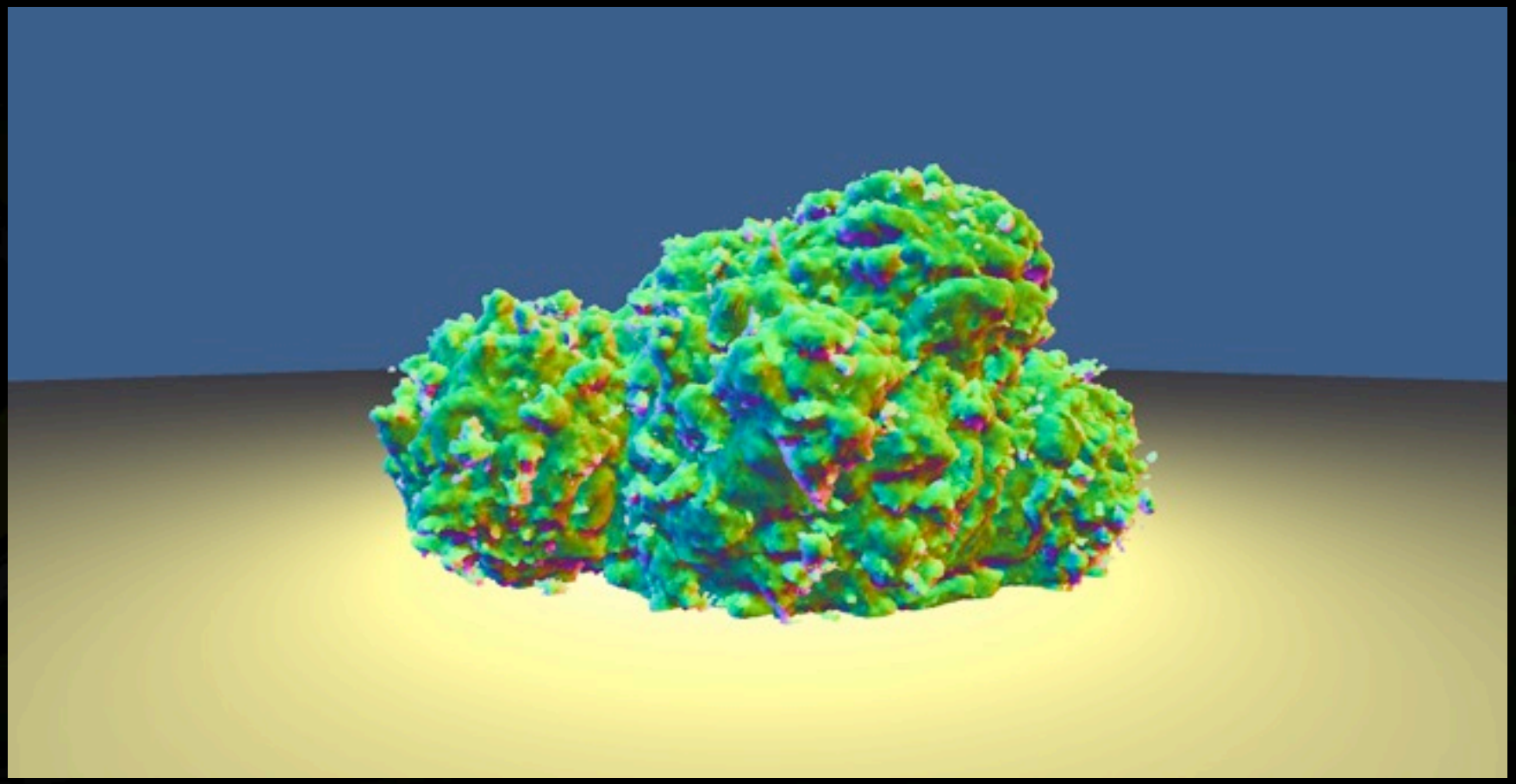
Add Displacement (Pyroclastic Noise)



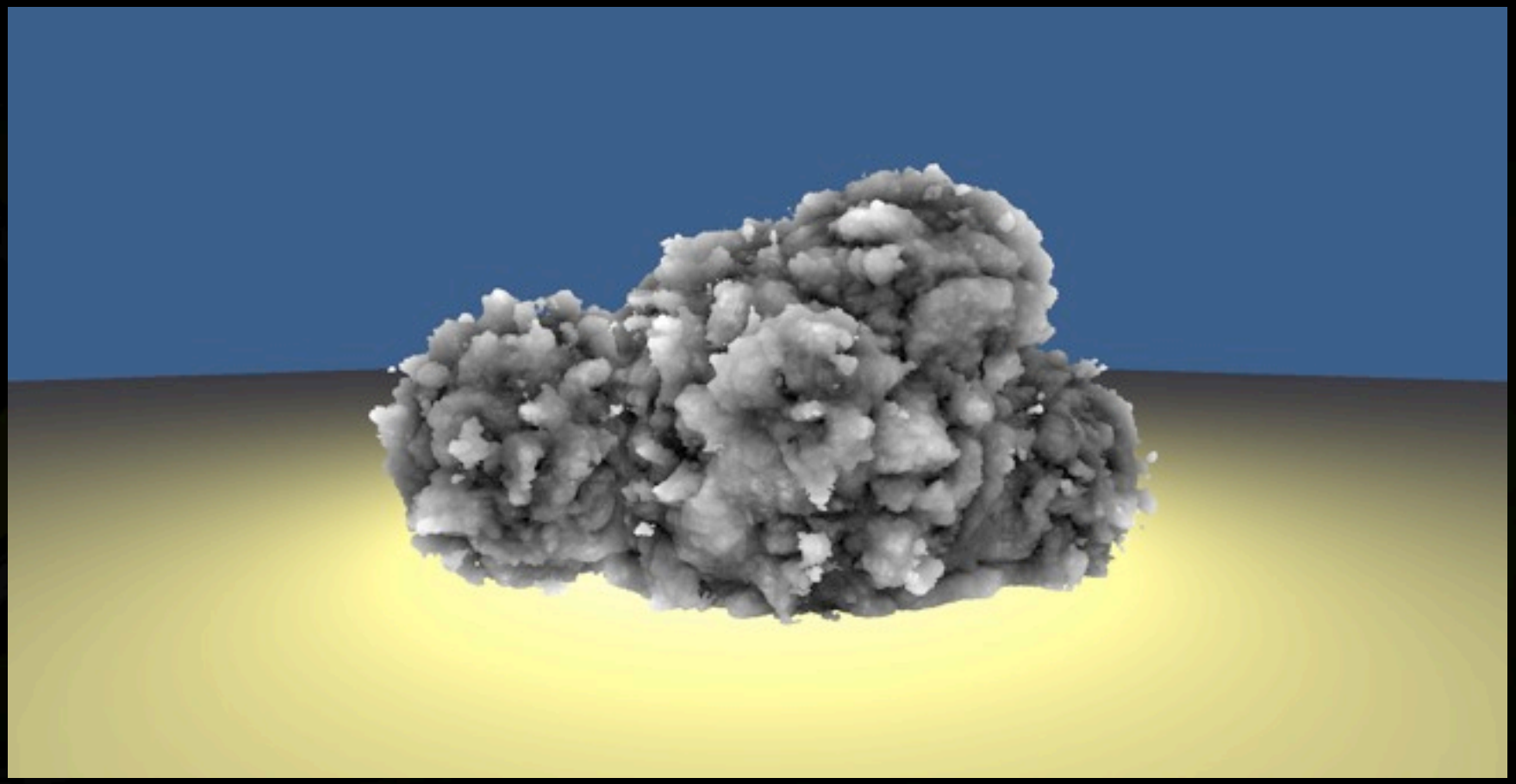
2 Octaves



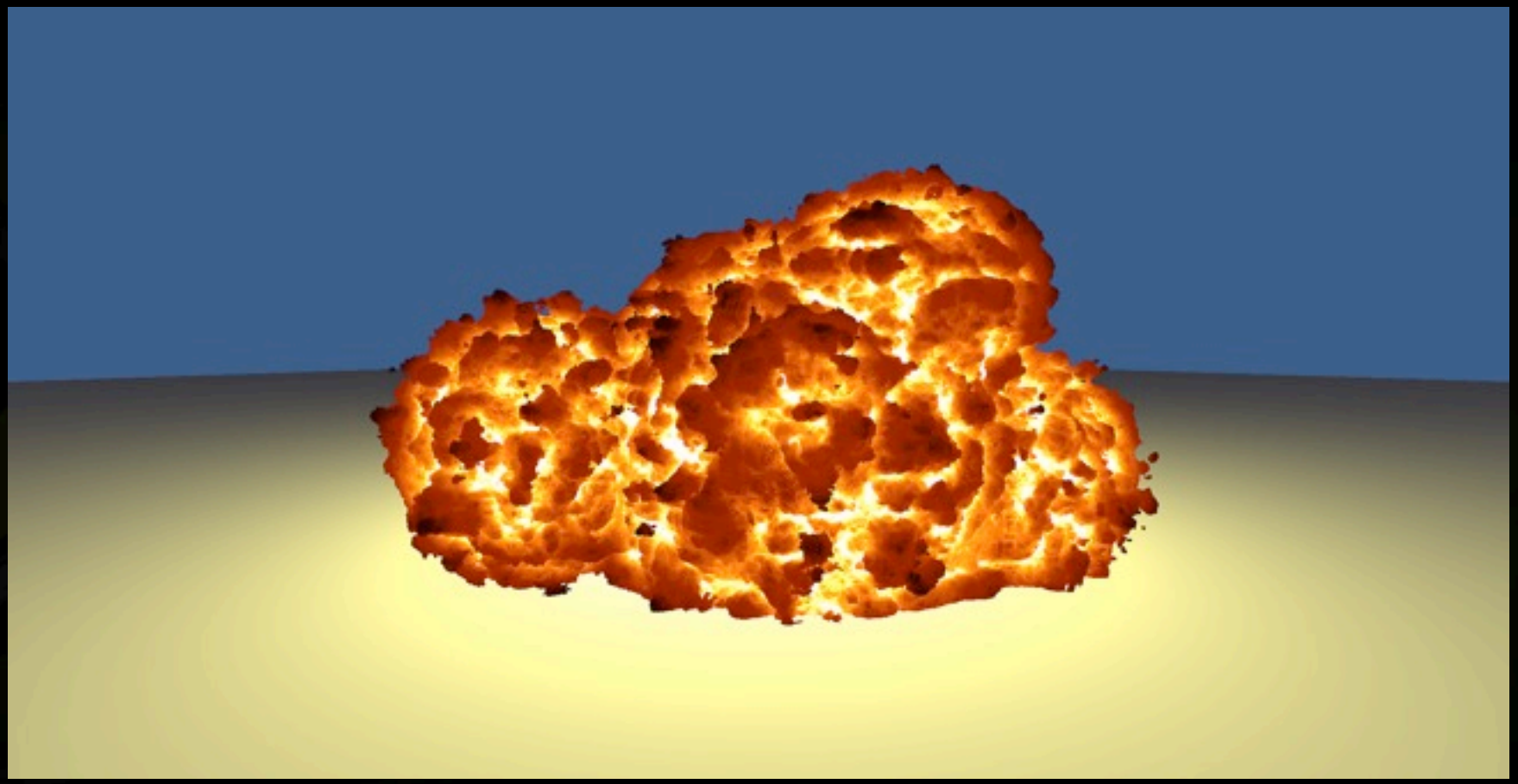
4 Octaves



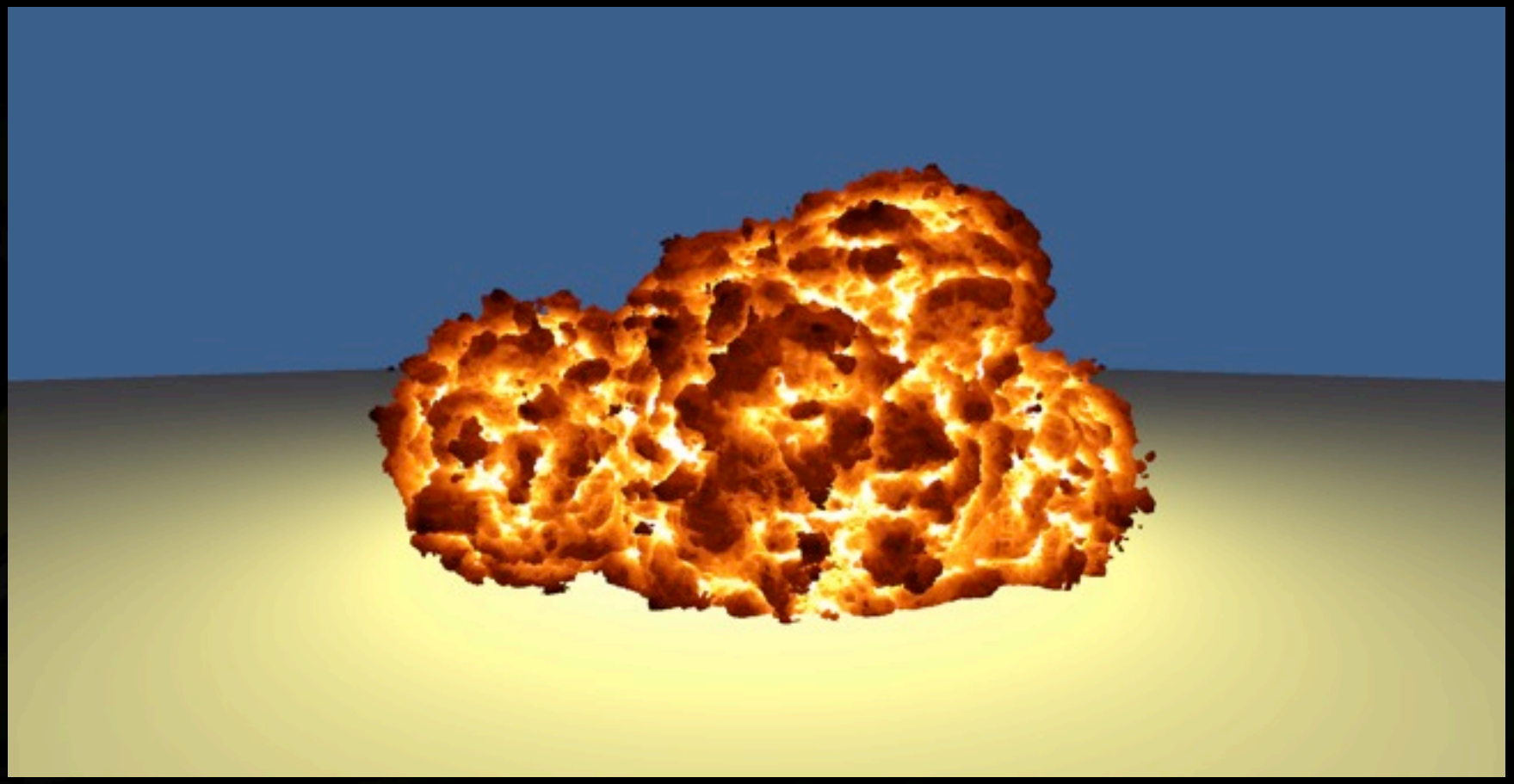
Shade Based On Displacement Distance



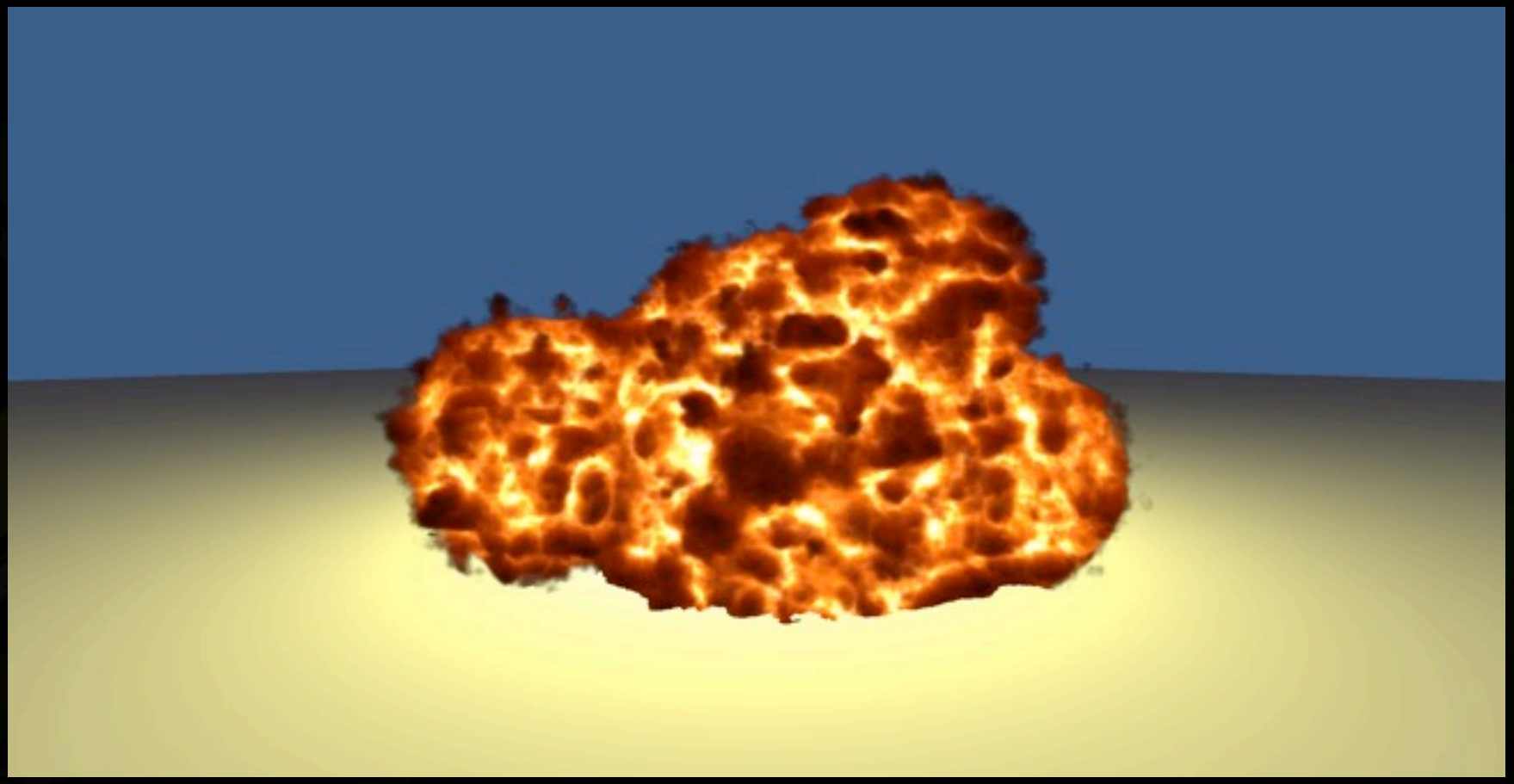
Add Color Gradient



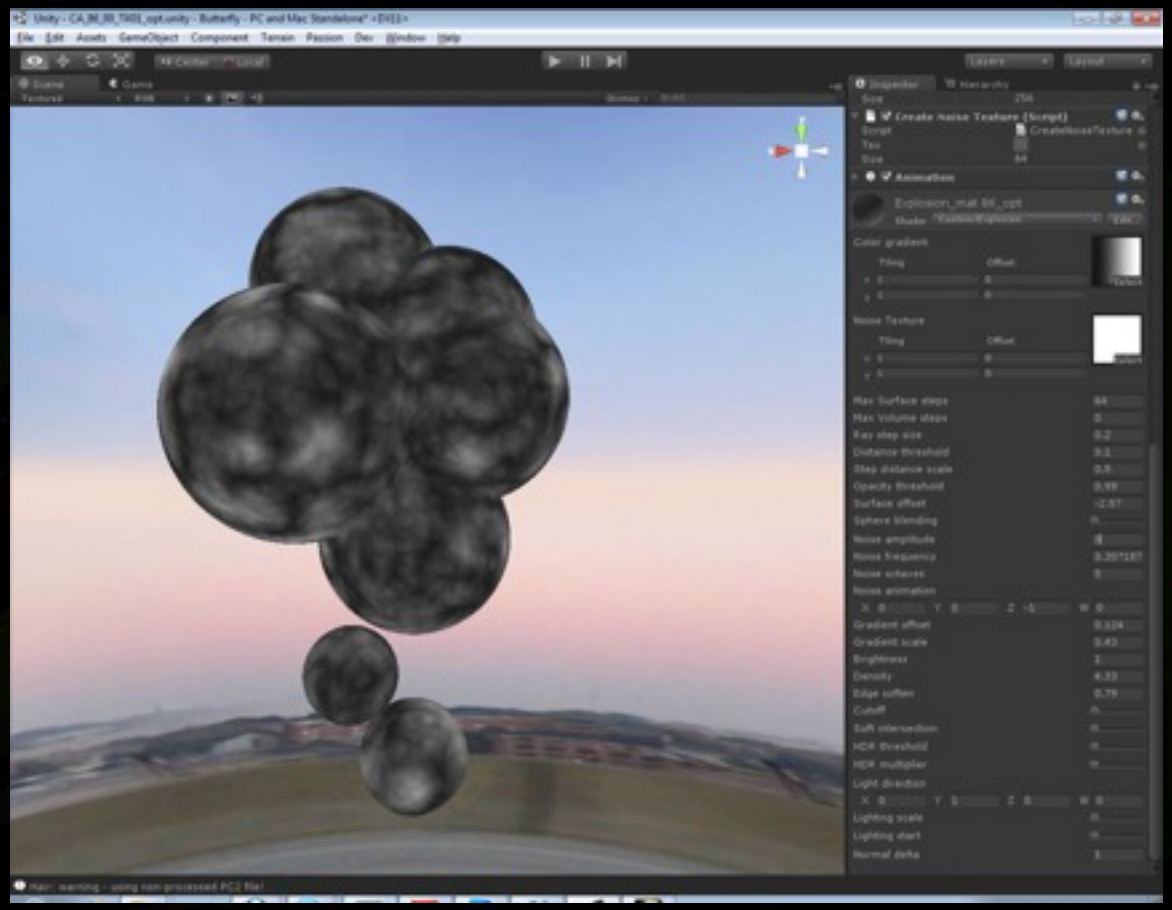
Add some lighting



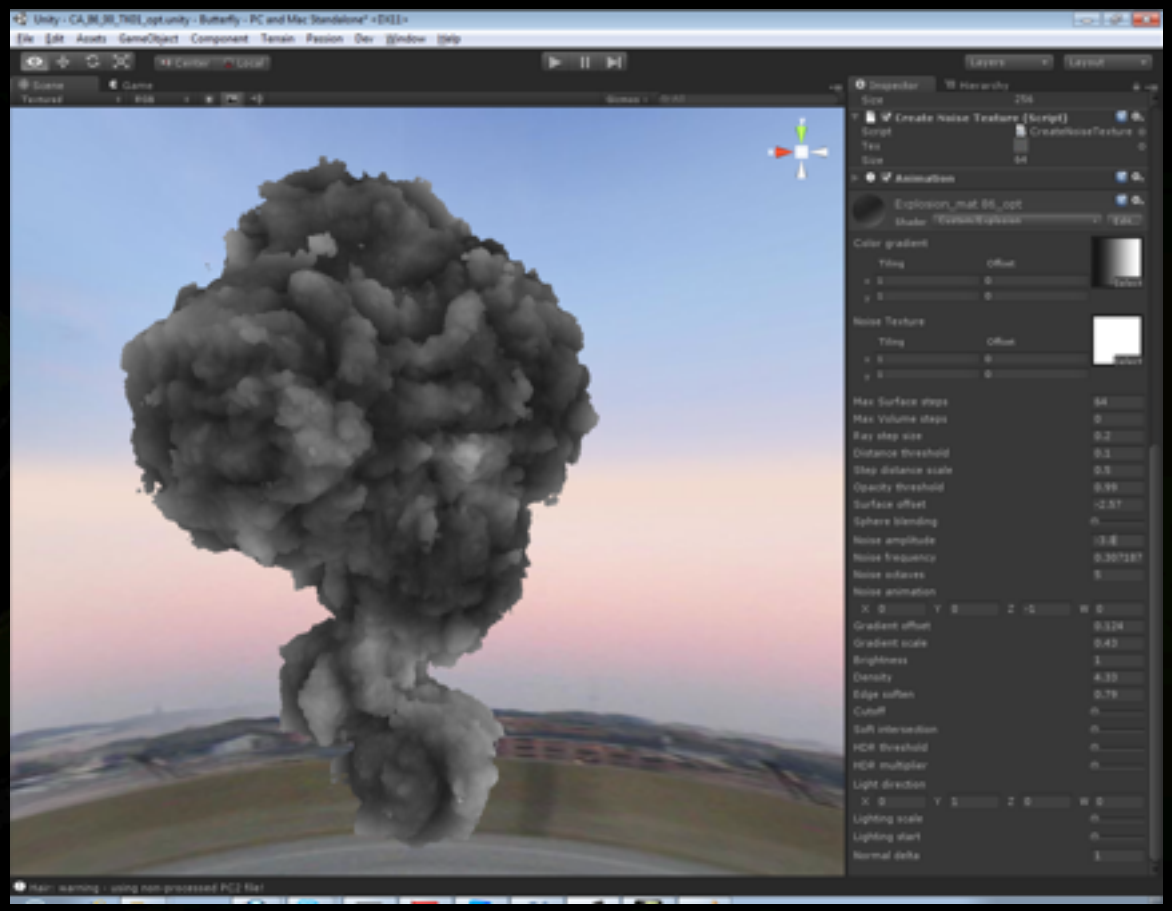
Add Volume Rendering



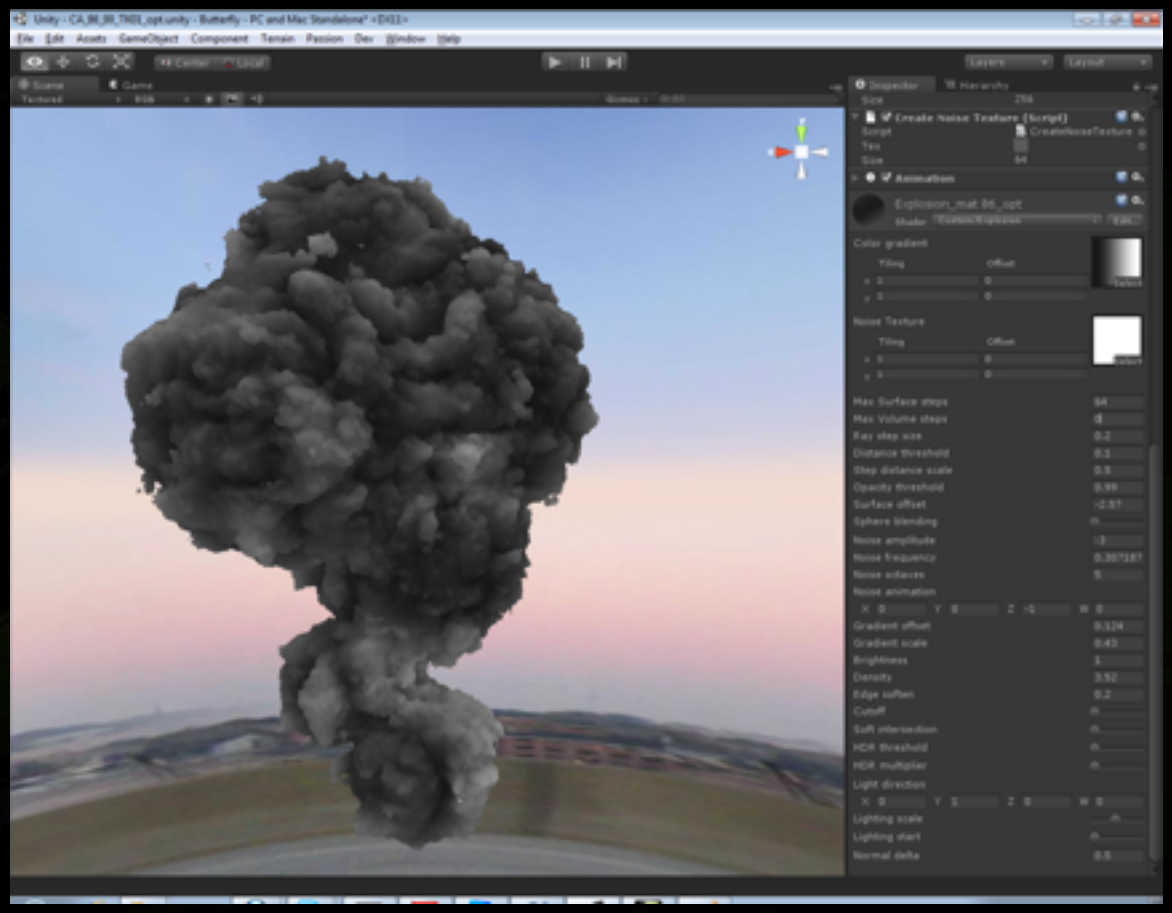
Smoke Stack



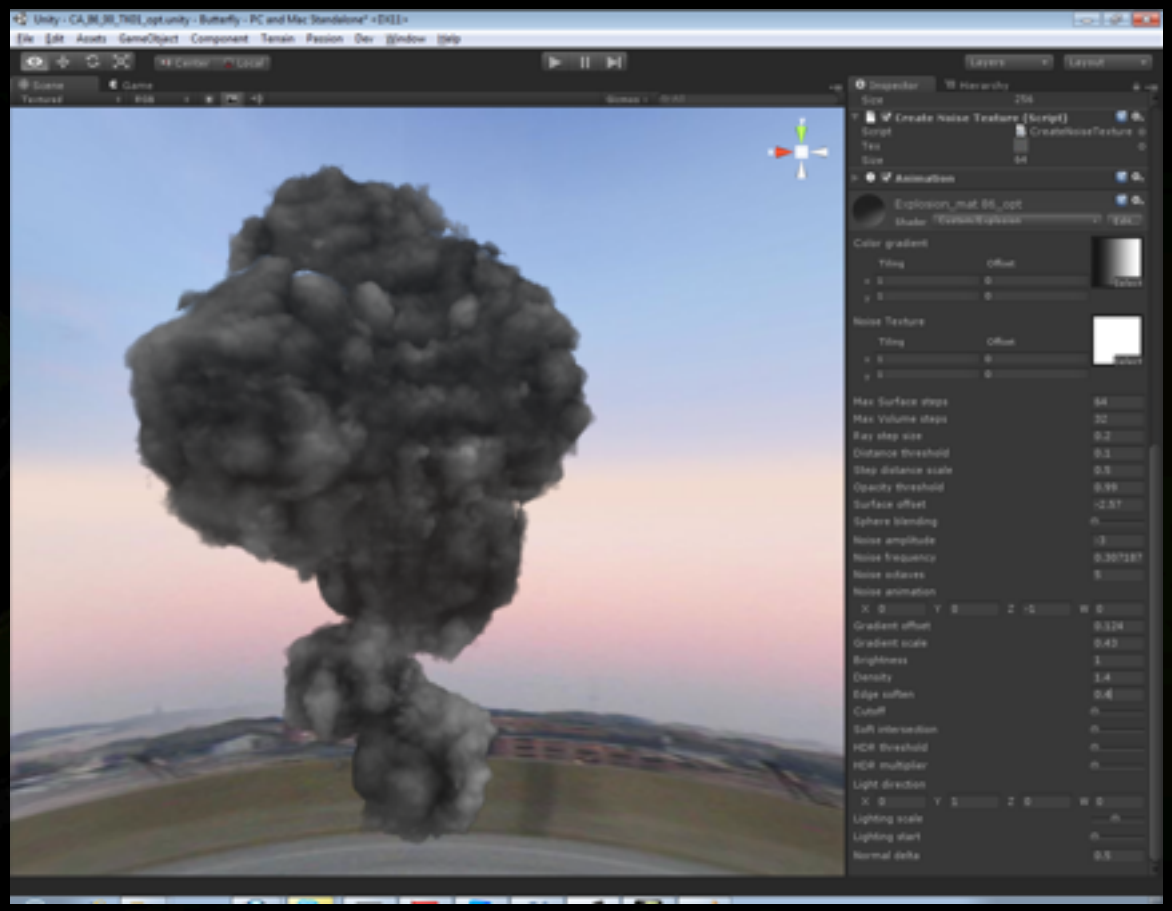
Smoke Stack



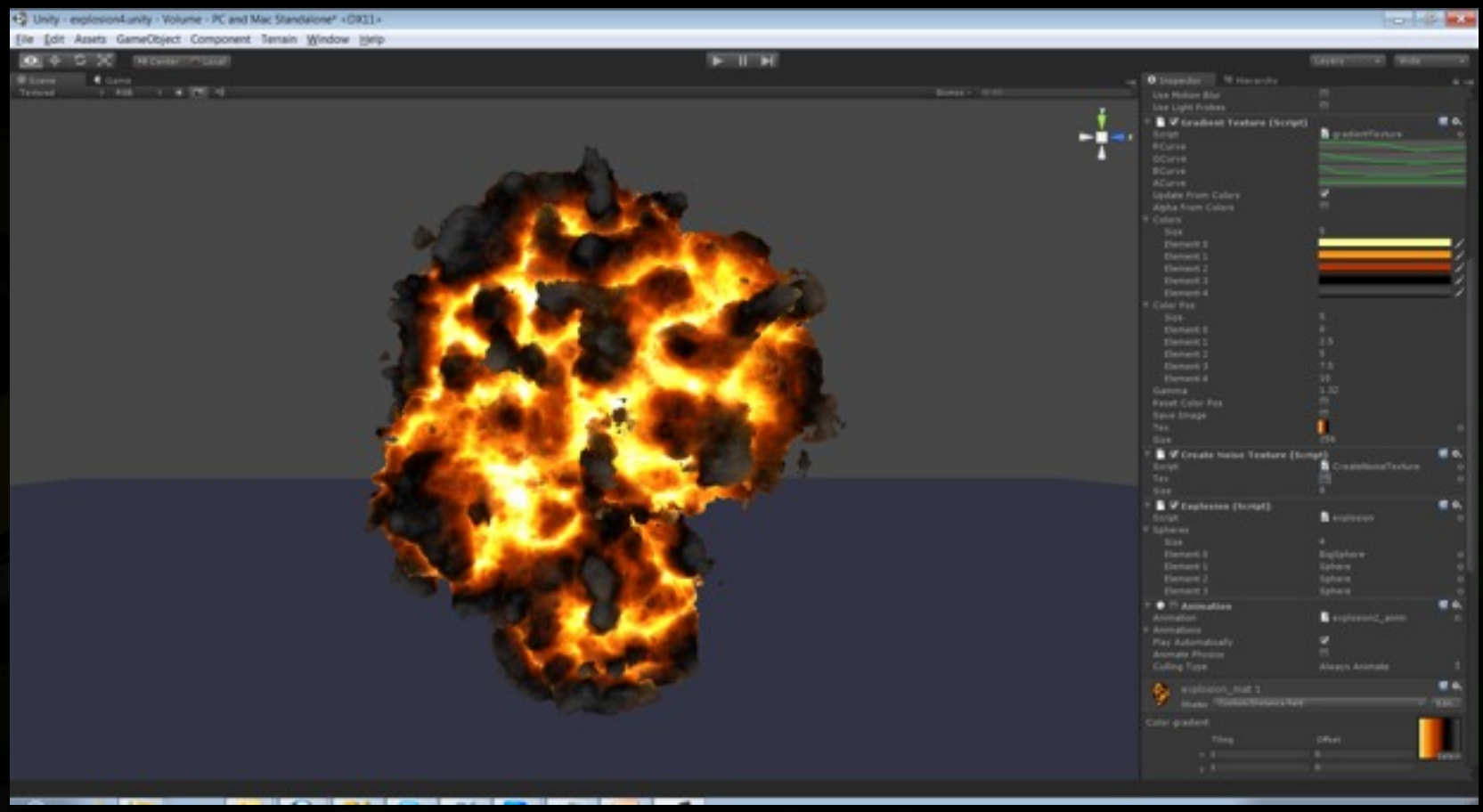
Smoke Stack



Smoke Stack



Another Example



Motion Blur

Motion Blur

- Improves “readability” in fast moving scenes
- Gives cinematic look
  - maybe this will change with movies moving to 48 fps and higher?
- Unity has an existing “motion blur” image effect
  - but not really motion blur
  - just leaves trails behind objects by blending new frames on top of old
- We wanted to implement a modern velocity-based motion blur in Unity

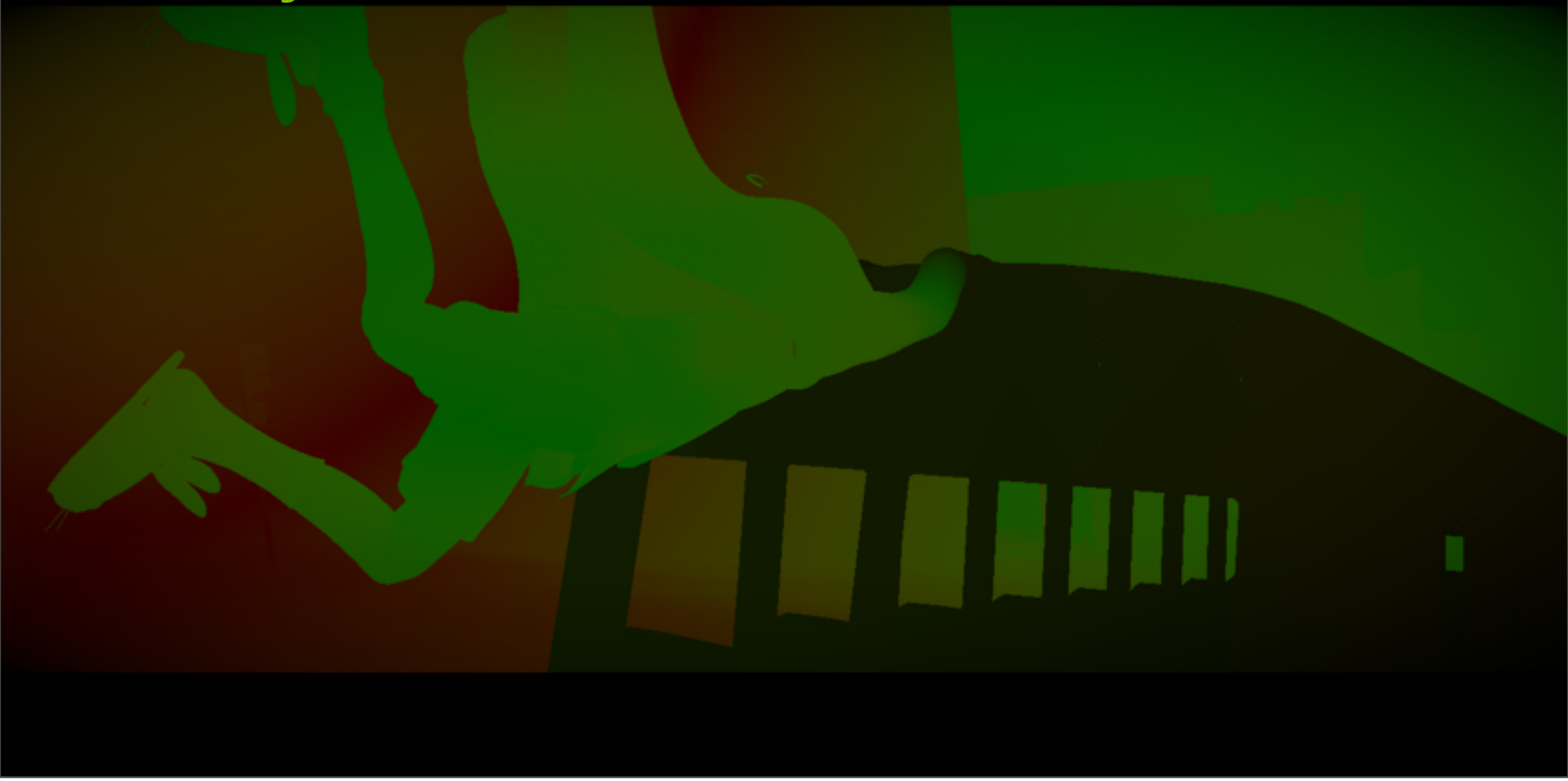
Velocity Buffer Motion Blur

- Generate velocity buffer
  - Shader calculates difference between previous and current positions in screen-space
  - Unity now provides
    - previous model matrix (UNITY_MATRIX_PREV_M)
    - Previous world space position for skinned objects (POSITION1)
  - Camera motion blur calculated from depth buffer and previous model view projection matrix
    - Have to be careful about motion blur on first frame of camera cuts!
- Use these velocities to blur an image of the current scene in direction of motion

Reconstruction Filter Motion Blur

- "A Reconstruction Filter for Plausible Motion Blur"
  - McGuire et al I3D 2012
- Uses current frame image and depth
  - Plus screen-space velocity buffer
- Simulates scatter as gather
  - accounts for occlusion using depth
  - blurs correctly outside object silhouettes
- Still has some artifacts
  - Especially at tile boundaries

Velocity Buffer Visualization



Motion Blur - Off

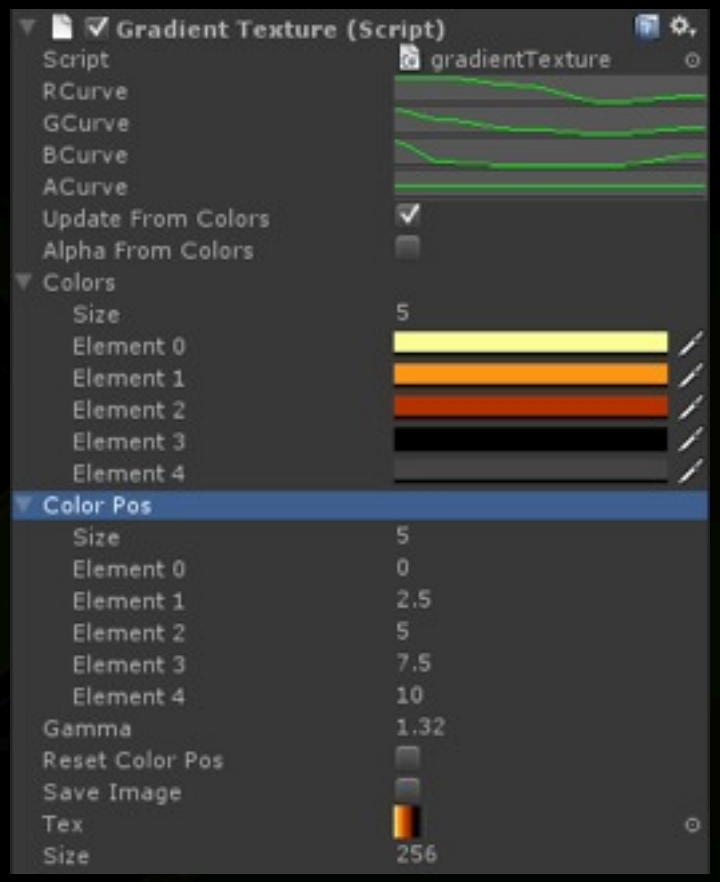
Simple Motion Blur

Reconstruction Filter Motion Blur

Things I Like About Unity

- Nice editor
- Very customizable
- Very fast to prototype new effects
- Automatically generates sliders for shader parameters
- Shader parameters can be animated over time using curves
- C# is fine once you get used to it!
  - like Java without the stupid bits
- Helpful developers and community…

Example: Gradient Editor script



Conclusion

- DirectX 11 brings state-of-the-art graphics to Unity
- Techniques from the off-line CG world are becoming possible in real-time
- Aim to make Unity as artist-friendly as possible

Thanks

- Everyone at Passion Pictures
  - Quentin Vien
  - Jamie Franks
  - Julian Hodgson
- Erland Körner
- Aras Pranckevičius
- Ole Ciliox
- Robert and Kuba Cupisz

Questions?

References

- Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces
  - John C. Hart, 1994
- Volume Rendering for Games
  - Simon Green GDC 2005
- Rendering Worlds with Two Triangles
  - Iñigo Quilez, NVSCENE 08
- Production Volume Rendering
  - Magnus Wrenninge et al, Siggraph 2011

References (cont.)

- Stupid OpenGL Shader Tricks
  - Simon Green, GDC 2003
- A Reconstruction Filter for Plausible Motion Blur
  - Morgan McGuire, Padraic Hennessy, Michael Bukowski, Brian Osman, I3D 2012