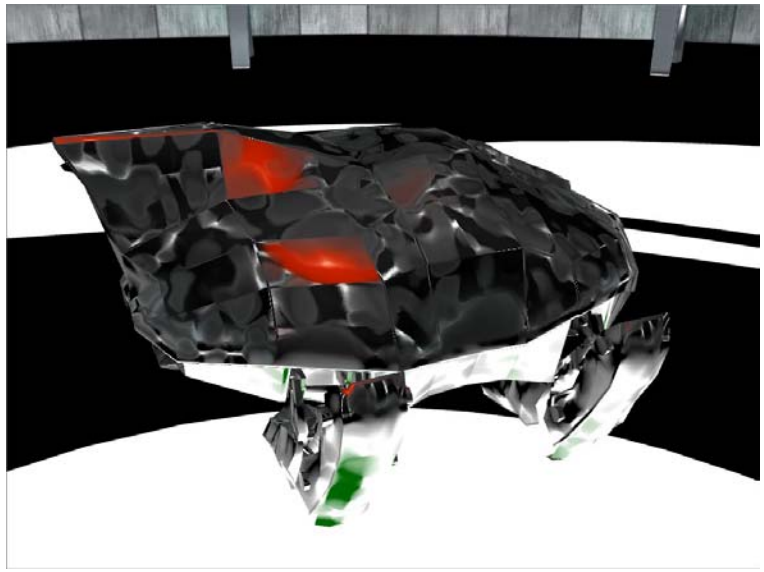
Summed-Area Tables
And Their Application to Dynamic Glossy Environment Reflections

Thorsten Scheuermann
3D Application Research Group

Overview

- Presenting work started by Justin Hensley, Ph.D. student at UNC and a 2004 ATI Fellowship recipient
- Summed-area tables
  - Use for blurring
  - Efficient creation
- Rendering dynamic reflections with per-pixel glossiness using dual-paraboloid maps and summed-area tables



Summed-Area Tables (SATs)

- Each element s_{mn} of a summed-area table S contains the sum of all elements above and to the left of the original table/texture T [Crow84]

$$s_{mn} = \sum_{i=1}^{m} \sum_{j=1}^{n} t_{ij}$$

	1	2	3	4
1	2	3	2	1
2	3	0	1	2
3	1	3	1	0
4	1	4	2	2

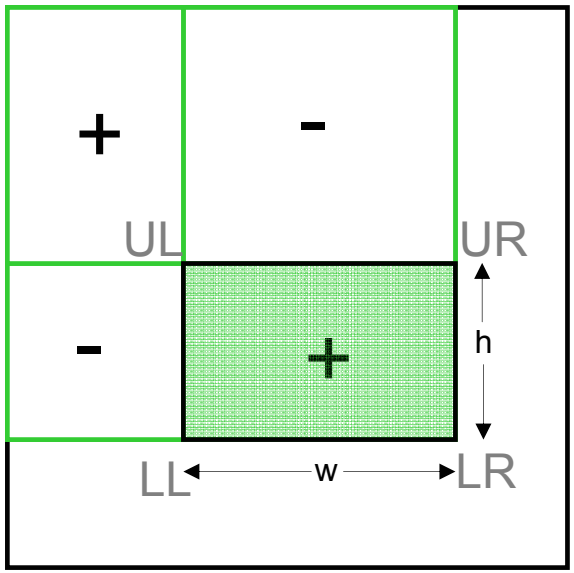
Original

2	5	7	8
5	8	11	14
6	12	16	19
7	17	23	28

Summed-area table

Using a Summed-Area Table

> Summed-area tables allow filtering the original texture with an arbitrary box filter in constant time



$$average = \frac{LR - UR - LL + UL}{w * h}$$

Filtering Example Code

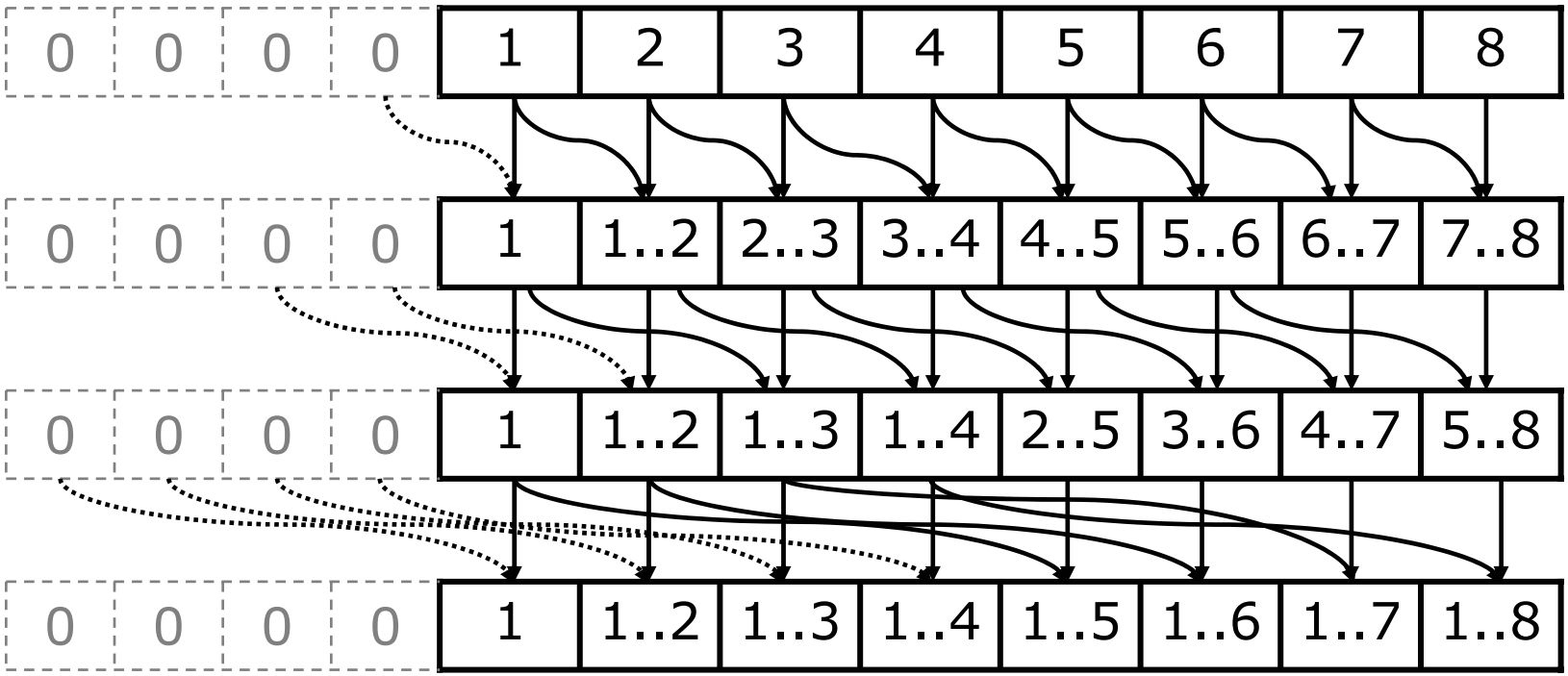
```
float4 tex2D_SAT_blur(sampler tSAT, float2 uv, float2 size)
{
  float4 result = tex2D(tSAT, uv + 0.5 * size);           // LR
  result -= tex2D(tSAT, uv + float2(0.5, -0.5) * size); // UR
  result -= tex2D(tSAT, uv + float2(-0.5, 0.5) * size); // LL
  result += tex2D(tSAT, uv - 0.5 * size);                 // UL
  result /= size.x * size.y;

  return result;
}
```

Efficient Summed-Area Table Creation

- Summed-area table construction can be decomposed into horizontal and vertical phase
- Each phase consists of $log_2(texture\ size)$ passes
- Each pass adds two elements from the previous pass
- Ping-pong between two rendertargets for each pass
- Horizontal Phase:

$$P_i(x,y) = P_{i-1}(x,y) + P_{i-1}(x - 2^{passindex}, y)$$

Horizontal Phase Example



Sampling off the texture returns 0 so that the sum doesn't get affected.

Saving Render Passes

- Adding two samples per pass requires $2*log_2(256) = 16$ passes for a 256x256 texture
- To reduce number of passes we can add more samples per pass
- Pass count reduces to $2*log_{\#samples}(texture\ size)$
- Using 16 samples per pass we only need 4 passes for converting a 256x256 texture to a summed-area table
  - Converting 512x512 needs 6 passes but two passes only need to add two texture samples

SAT Creation Vertex Shaders

```
float fPassIndex;
float2 vPixelSize;
float4x4 mVP;

struct VsOutput
{
   float4 pos        : POSITION0;
   float4 uv[8]      : TEXCOORD0; // 16 UVs stuffed in 8 float4's
};

VsOutput main(float4 inPos: POSITION, float2 inUV : TEXCOORD0)
{
   VsOutput o;

   // transform vertex (assuming app has set up screen-aligned
   // quad drawing)
   o.pos = mul (inPos, mVP);

[…]
```

SAT Creation Vertex Shader

```
[…]
  float passOffset = pow(16.0, fPassIndex) * vPixelSize.x;

  // output first two texcoords
  o.uv[0].xy = inUV;
  o.uv[0].wz = o.uv[0].xy - float2(passOffset, 0);

  // compute remaining 14 texcoords for neighboring pixels
  for (int i=1; i<8; i++) {
    o.uv[i].xy = o.uv[0].xy -
                 float2((2.0 * i) * passOffset, 0);
    o.uv[i].wz = o.uv[0].xy -
                 float2((2.0 * i + 1.0) * passOffset, 0);
  }

  return o;
}
```

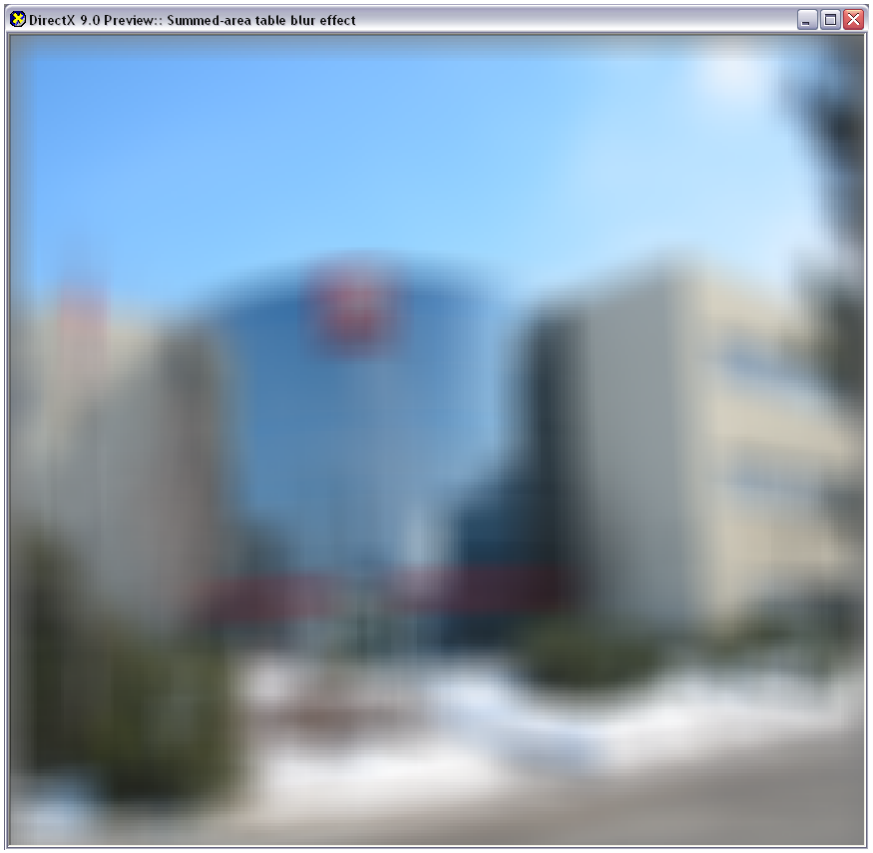
SAT Creation Pixel Shader

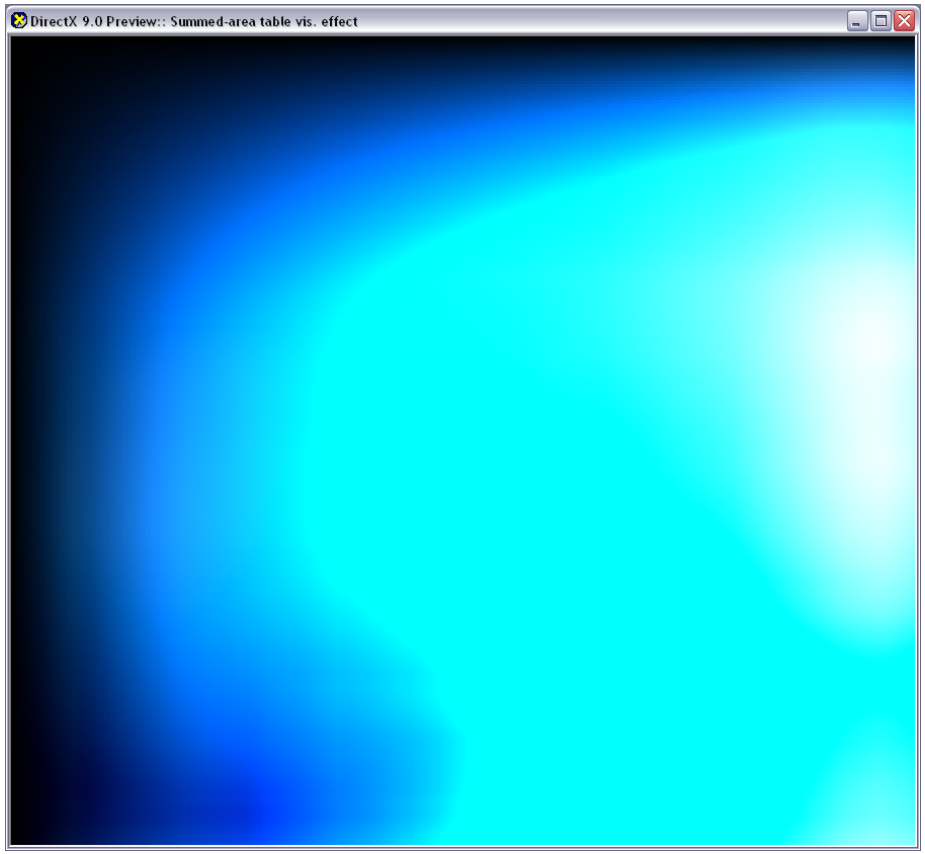
```
float4 SATPass(sampler tSrc, float4 uv[8])
{
  float4 t[8];

  // add 16 texture samples with pyramidal scheme
  // to maintain precision
  for (int i=0; i<8; i++) {
    t[i] = tex2D(tSrc, uv[i].xy) +
           tex2D(tSrc, uv[i].wz);
  }

  t[0] += t[1]; t[2] += t[3];
  t[4] += t[5]; t[6] += t[7];

  t[0] += t[2]; t[4] += t[6];

  return t[0]+t[4];
}
```

Precision Requirements

- For proper reconstruction a summed-area table needs $log_2(texture\ width)+log_2(texture\ height)+bit\ depth\ of\ source\ texture$ bits of precision
- Use float32-rendertargets to compute and store summed-area tables
- Precision errors are unbiased and average out as you use larger box filter kernels

Summed-Area Table Example





Boundary Conditions

- To make sure sampling off the texture does not mess up the results we need to set up the correct texture clamping behavior
- Two possibilities:
  - Clamp to border color with a color of (0, 0, 0, 0)
  - Render a black border around the texture to be converted into SAT and set Clamp to Edge mode

SAT Precision Improvement Trick

- Bias input texture by -0.5 before generating summed-area table
- When reconstructing samples from SAT, undo bias by adding 0.5 to final result
  - This helps because now we take advantage of the sign bit
- Even better: Use average color of input as bias (instead of 0.5)
- Improving precision of summed-area tables is particularly useful when using hardware with limited pixel pipeline precision

No precision improvement

With precision improvement

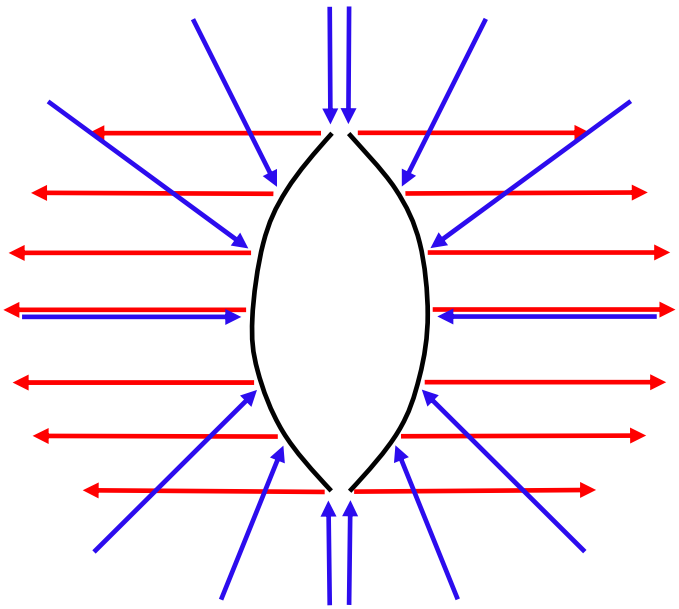
Dynamic Glossy Reflections Game Plan

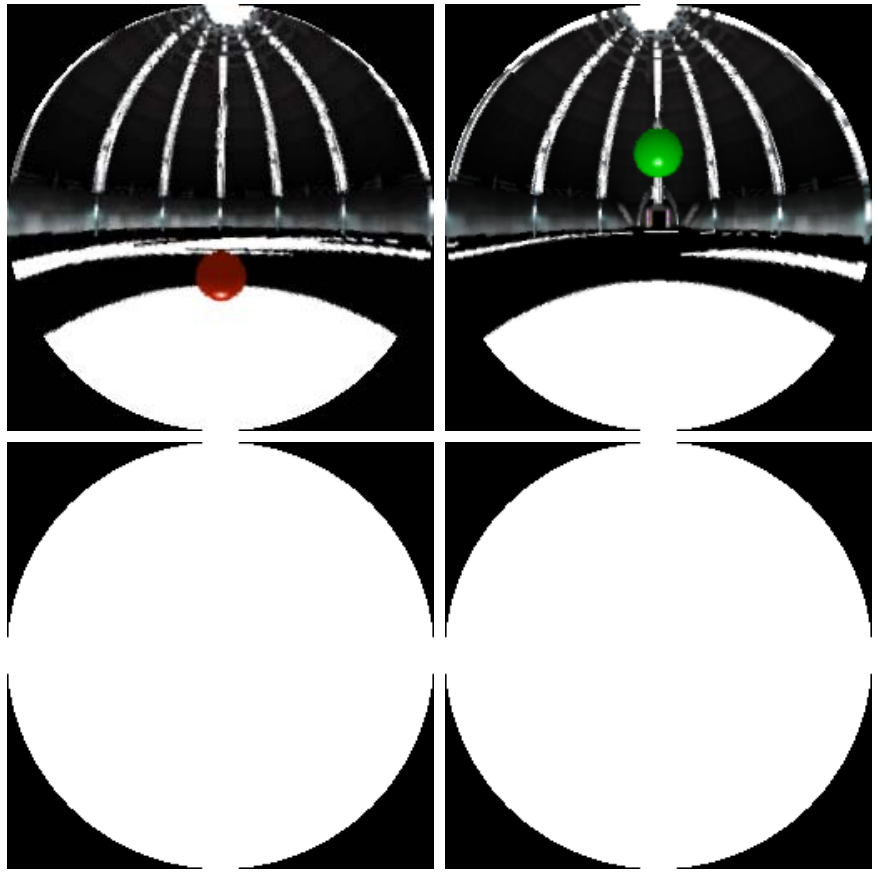
- Render dynamic cubemap
- Convert to dual-paraboloid map
- Convert dual-paraboloid map faces to summed-area tables
- Apply SAT DP-map to glossy object

- Sounds like a lot of work, but is actually quite fast on modern hardware
  - Will show real-time demo in a minute

The Return of the Dual-Paraboloid Map!

- Dual-paraboloid map = two textures that store an environment as reflected in parabolic mirrors





Color channels

Alpha channel

DP Sampling Shader

```
float3 texDP (sampler tFront, sampler tBack, float3 dir)
{
  // convert 3D lookup vector into 2D texture coordinates
  float2 frontUV = float2(0.5, -0.5) * dir.xy /
                   (1.0 - dir.z) + 0.5;
  float2 backUV =  float2(0.5, -0.5) * dir.xy /
                   (1.0 + dir.z) + 0.5;

  // sample DP map faces and blend together
  float4 cFront = tex2D (tFront, frontUV);
  float4 cBack = tex2D (tBack, backUV);

  return cFront.rgb + cBack.rgb;
}
```

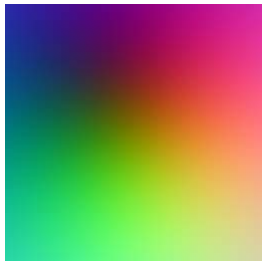
Cubemap to DP Map Conversion

- Convert uv position on DP map face to 3D vector using these equations: (from [Blythe99])

Front face: $$R = \begin{pmatrix} \frac{2u}{u^2+v^2+1} \\ \frac{2v}{u^2+v^2+1} \\ \frac{-1+u^2+v^2}{v^2+v^2+1} \end{pmatrix}$$

Back face: $$R = \begin{pmatrix} \frac{-2u}{u^2+v^2+1} \\ \frac{-2v}{u^2+v^2+1} \\ \frac{1-u^2-v^2}{u^2+v^2+1} \end{pmatrix}$$

- Perform cubemap lookup and store result in DP face
- Precompute lookup textures or do math on the fly



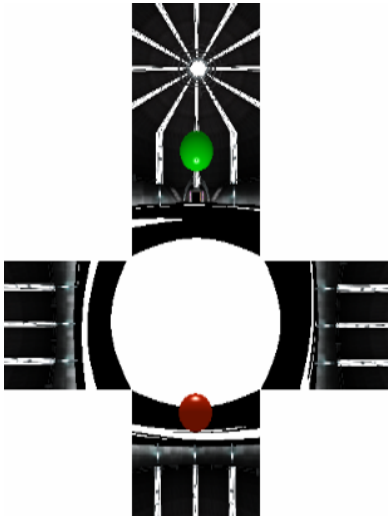
Front lookup texture

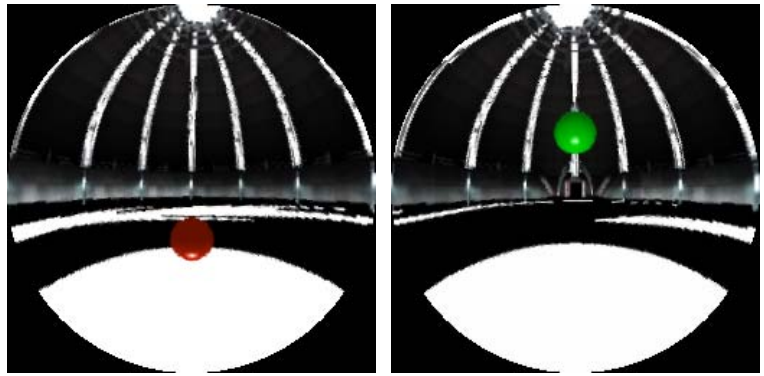
Back lookup texture

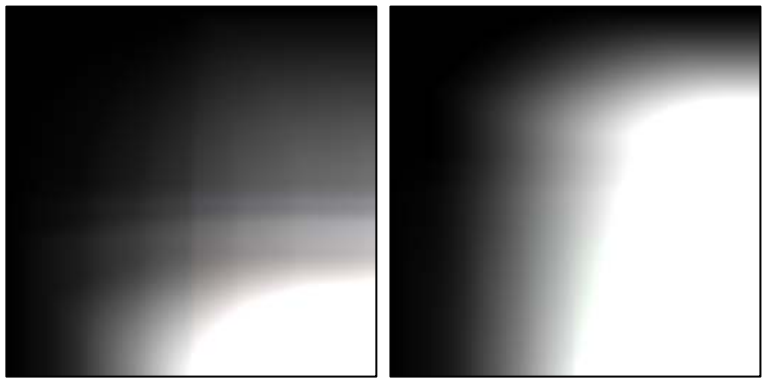
Why Bother With DP Mapping?

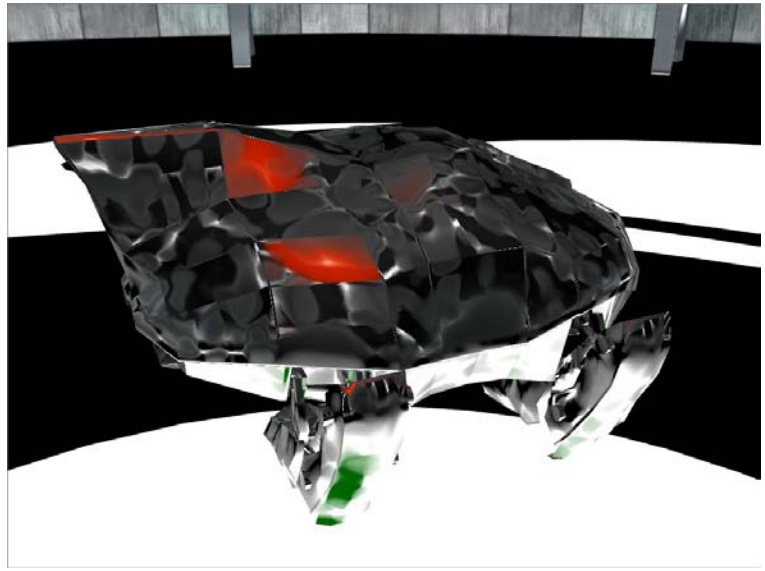
- Summed-area table concept does not map to cubemaps which are in spherical domain
  - Summed-area tables only work on rectangular 2D images
- Filtering with a box filter in dual-paraboloid space causes less distortion

Putting it All Together







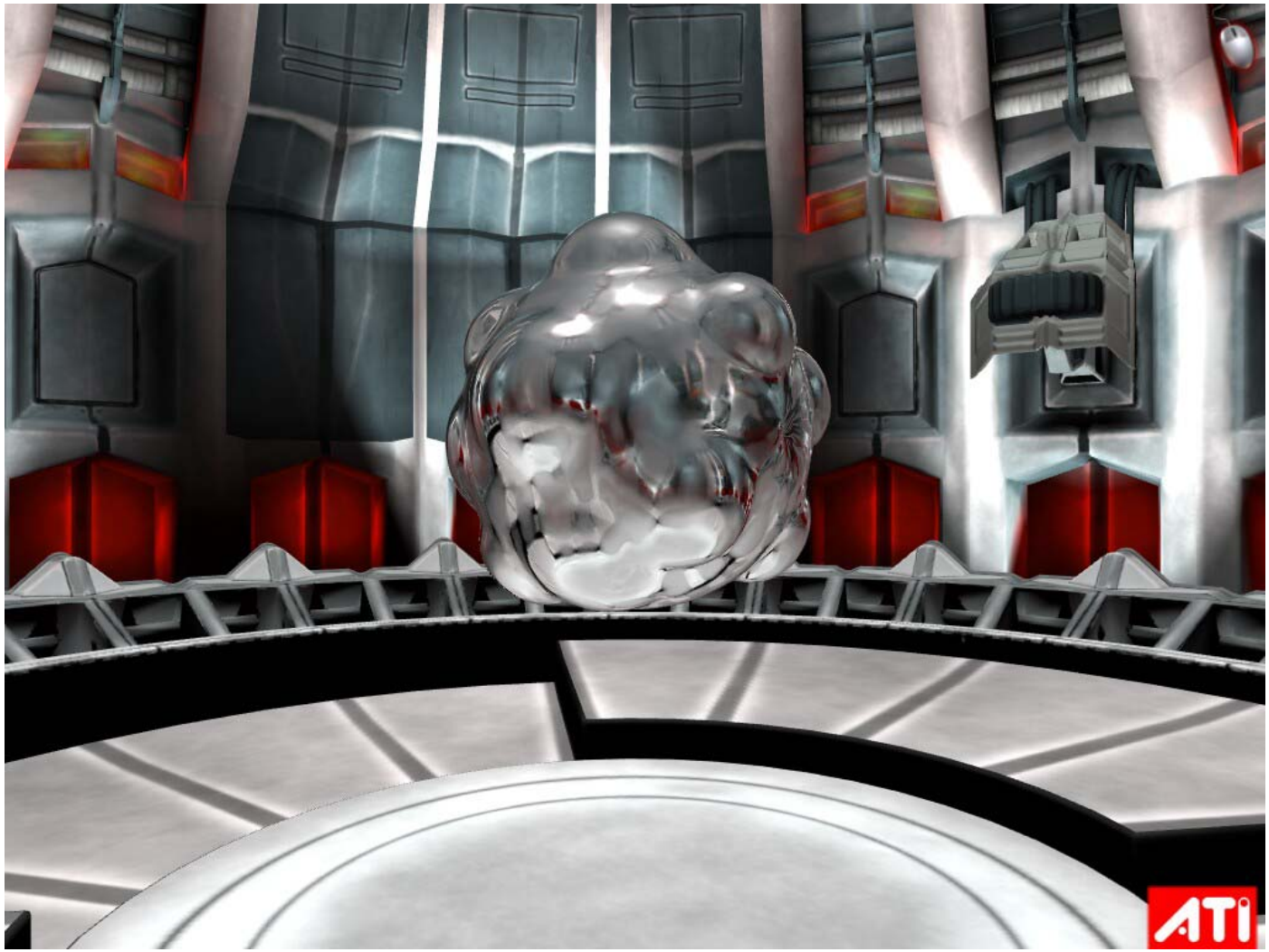


SAT DP Sampling Shader

> Very similar to DP sampling, but uses SAT texture lookup function:

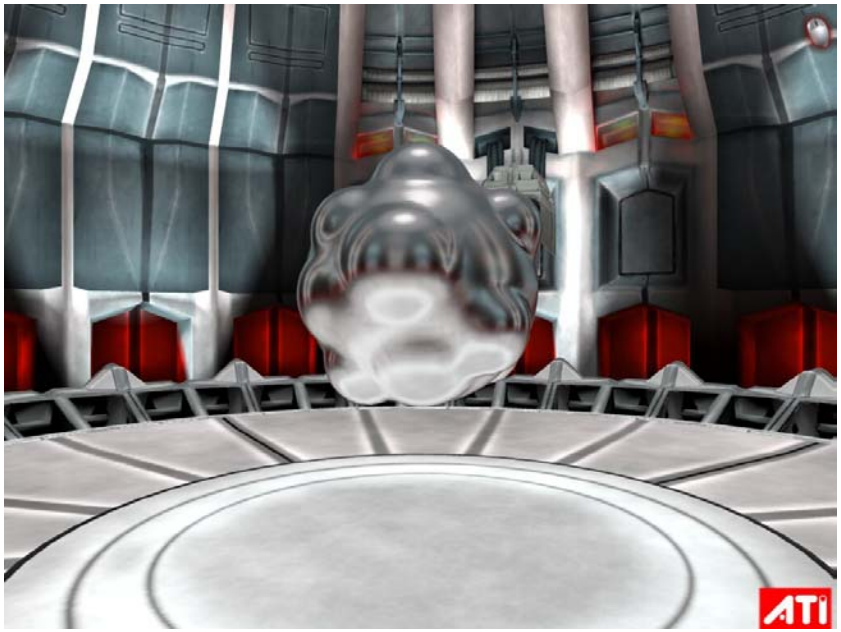
```
float3 texDP_SAT (sampler tFront, sampler tBack, float3 dir,
                  float2 filterSize)
{
  // convert 3D lookup vector into 2D texture coordinates
  float2 frontUV = float2(0.5, -0.5) * dir.xy /
                   (1.0 - dir.z) + 0.5;
  float2 backUV =  float2(0.5, -0.5) * dir.xy /
                   (1.0 + dir.z) + 0.5;

  // sample DP map faces and blend together
  float4 cFront = tex2D_SAT_blur (tFront, frontUV, filterSize);
  float4 cBack = tex2D_SAT_blur (tBack, backUV, filterSize);
  float4 cRefl = cFront + cBack;

  // normalize result
  return cRefl.rgb / cRefl.a;
}
```

Demo



Other Possibilities

- Average several box-filtered environment map samples to approximate smoother blur filter kernels
- Approximate Phong BRDF by combining very blurred sample in normal direction and less blurry sample in reflection vector direction for specular lobe



Direct DP Face Rendering

- Alternative to rendering cubemap, then converting to DP map:
- Transform environment using parabolic projection function and render directly into DP faces
- Unfortunately parabolic projection is non-linear and maps lines to curves
  - Linear rasterization of graphics hardware causes artifacts
  - Might be OK if your geometry is tesselated highly enough
- See [Coombe04] for details

Disadvantages

- Precision requirements for summed-area tables
- float32 textures used for summed-area tables do not currently support bilinear filtering
  - Not so much of an issue when you blur enough
  - Can perform bilinear filtering manually in the shader

Conclusion

- Using summed-area tables for blurring
- Efficient summed-area table generation scheme
- Converting cubemap into dual-paraboloid maps
- Using summed-area tables and dual-paraboloid mapping together to achieve dynamic glossy environment reflections

Acknowledgements

- Justin Hensley
- Thanks to Eli Turner for the demo artwork

References

- [Crow84] Crow, F. C., *Summed-area tables for texture mapping*. Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques ACM Press: 207-212 1984
- [Coombe04] Coombe, G., Harris, M. and Lastra, A., *Radiosity on Graphics Hardware*. Graphics Interfaces, 2004.
- [Blythe99] Blythe, D., *Advanced Graphics Programming Techniques Using OpenGL*. SIGGRAPH 1999 course notes.
  http://www.opengl.org/resources/tutorials/sig99/advanced99/notes/node184.html