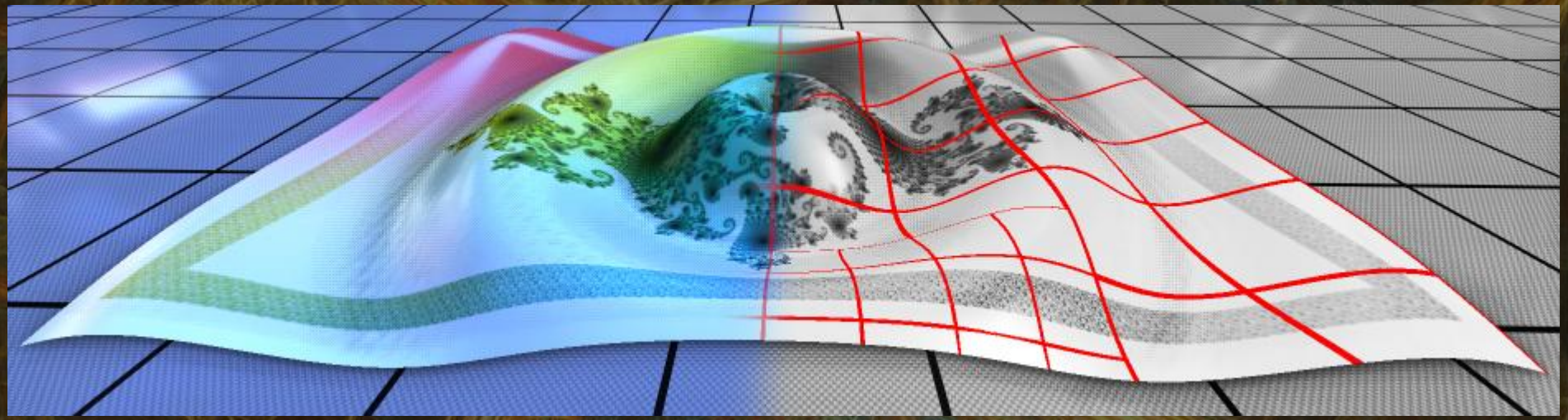

SIGGRAPH2008

Advanced Virtual Texture Topics



Martin Mittring

Lead Graphics Programmer

Presentation Overview

- Motivation
- Virtual Texture*
  - Implementation
  - Related Topics
- Combo Texture*
- Summary / Future

* Demo

Motivation: Texture Streaming

Peripherals	Computer	GPU
Hard drive	Main memory	GPU texture cache
Network		Texture
DVD/CD/…		Video memory

Amount ⟷ Speed

Per mip-map texture streaming*

- Streaming is needed:
  - Large worlds, Loading time, Memory limits
- HW / API support
  - No asynchronous single mip-map update
  - Create/Release breaks MultiGPU
  - Unpredictable performance
- less stable performance

* Used in Crysis™ to overcome 32 bit limits with high resolution textures

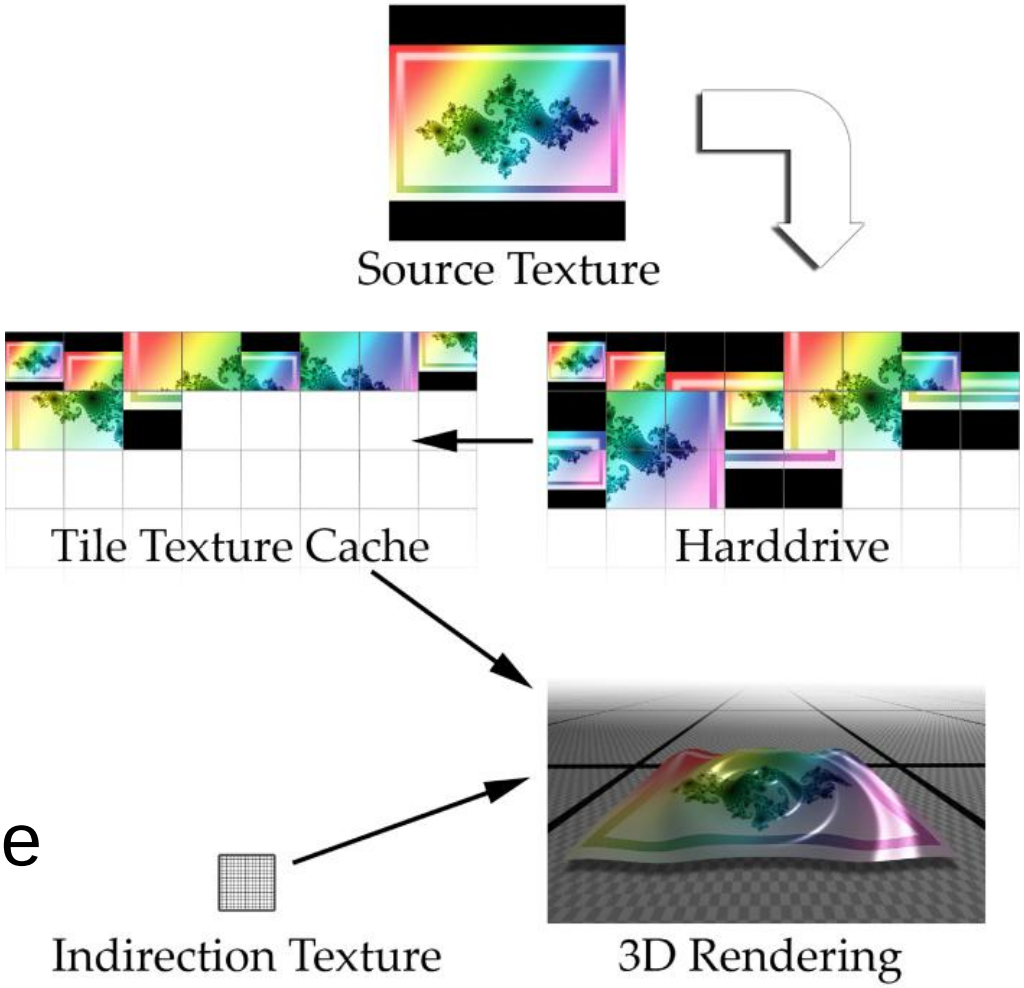
What is a Virtual Texture*?

- Emulates a mip-mapped texture
  - can be high resolution
  - real-time on consumer graphics hardware
  - partly resides in texture memory
- Implications on engine design, content creation, performance and image quality

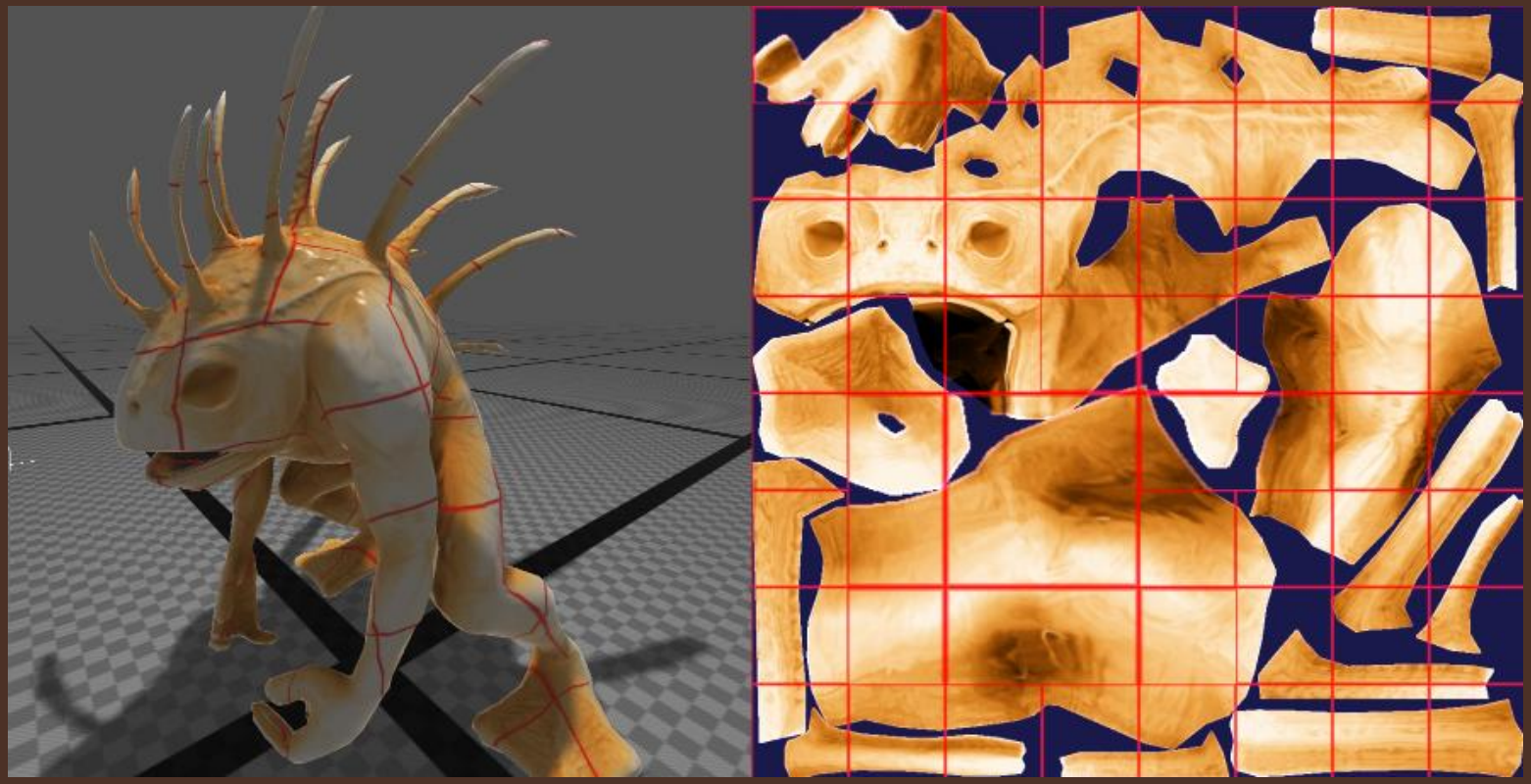
* “Virtual Texture” derived from the OS/CPU feature “Virtual Memory”

Virtual Texture example usage

- Prepare mip-maps, store in equal sized tiles on HD
- Compute required tiles and request from HD
- Update indirection texture and tile cache
- Render 3D scene



Virtual Texture Demo



Virtual Texture in the Pixel Shader

- Idea:

  single unfiltered lookup into indirection texture (scale&offset),
  single filtered lookup into tile cache texture
  -> GDC 2008 Sean Barrett “Sparse Virtual Textures”
- No mips, memory coherent access
- Precision problem: 24/32bit float / integer
- D3D9: Half texel offset to get stable results
- Alternatives: Indirection per draw call, …

Virtual Texture Pixel Shader (HLSL)

```
float4 g_vIndir;            // w,h,1/w,1/h indirection texture extend
float4 g_Cache;             // w,h,1/w,1/h tilecachetexture extend
float4 g_CacheMulTilesize; // w,h,1/w,1/h tilecachetexture extend * tilesize

sampler IndirMap = sampler_state
{
  Texture    = <IndirTexture>;
  MipFilter = POINT;  MinFilter = POINT;  MagFilter = POINT;
//  MIPMAPLODBIAS = 7; // using mip-mapped indirection texture, here 128x128
};

float2 AdjustTexCoordforAT( float2 vTexIn )
{
  float fHalf = 0.5f;        // half texel for DX9, 0 for DX10
  float2 TileIntFrac = vTexIn*g_vIndir.xy;
  float2 TileFrac = frac(TileIntFrac)*g_vIndir.zw;
  float2 TileInt = vTexIn - TileFrac;
  float4 vTiledTextureData = tex2D(IndirMap,TileInt+fHalf*g_vIndir.zw);
  float2 vScale = vTiledTextureData.bb;
  float2 vOffset = vTiledTextureData.rg;
  float2 vWithinTile = frac( TileIntFrac * vScale );

  return vTileCacheUV = vOffset + vWithinTile*g_CacheMulTilesize.zw
         + fHalf*g_Cache.zw;
}
```

Bilinear filtering

- Efficient only with redundant border
- 1 is minimum, DXT 4 pixel (e.g. 128+4), centered tiles can improve quality
- Texture updates become unaligned and texture size non power-of-two
- Power-of-two property can be retained by reducing the usable tile size but quality suffers (e.g. 124+4)

Indirection Texture

- Efficient representation of the dynamic (view dependent) Quad-tree
- Texture format:
  - 32Bit ARGB (Precision issues on some HW)
  - 64Bit FP16 for precision and less PS math
- Free channel can be used to fade tiles: bilinear filtered, to limit max per-pixel LOD

Indirection Texture Update

- Quad-tree modifications only at leaf level
- 2D block updates of the texture (CPU/GPU)
- Mip mapped texture offers per-pixel LOD
- Many indirection textures at full resolution in memory are wasteful
- Multiple indirection texture updates per frame

Indirection Texture Update (C/C++)

```
// float to fp16(s1e5m10) conversion (does not handle all cases)
WORD float2fp16( float x )
{
  uint32 dwFloat = *((uint32 *)&x);
  uint32 dwMantissa = dwFloat & 0x7fffff;
  int iExp = (int)((dwFloat>>23) & 0xff) - (int)0x7f;
  uint32 dwSign = dwFloat>>31;
  return (WORD)( (dwSign<<15)
      | (((uint32)(iExp+0xf))<<10)
      | (dwMantissa>>13) );
}

WORD texel[4];                          // texel output
RECT recTile;                           // in texels in the tilecache texture
int iLod;                               // 0=full domain, 1=2x2, 2=4x4, ...
int iSquareExtend;                      // indirection texture size in texels
float fInvTileCache;                    // tile.Width / texCacheTexture.Width

texel[0] = float2fp16(recTile.left*fInvTileCache);
texel[1] = float2fp16(recTile.top*fInvTileCache);
texel[2] = float2fp16((1<<iLod)/ iSquareExtend);

texel[3] = 0;                           // unused
```

Tile Texture Cache Update 1/2

- Requirements:
  - Fast in latency and throughput
  - stall free (no wait) but correct state
  - Minimal bandwidth and memory overhead
  - Memory layout and type for fast texture lookups
  - O(1) for one tile
  - Predictable performance

Tile Texture Cache Update 2/2

- Direct CPU Update

  D3D9: D3DPOOL_MANGED, LockRect()

- Small intermediate tile texture

  D3D9: LockRect(), StretchRect(), not for compressed

- Large intermediate tile cache texture

  D3D9: D3DPOOL_SYSTEM, LockRect(), UpdateTexture()

- GPU update (render to texture)

  fastest, flexibility limited

Limits

- Maximum virtual texture size

  $= \text{Extend}_{\text{Indirection texture}} * \text{Extend}_{\text{tile without border}}$

  (addtionally limited by math precision)

- Maximum tile cache texture size
- Storing different attributes in the tile caches
- Splitting the tile cache over multiple textures
  - Different object types or multiple cache levels

Possible Tile Sources

- Harddrive, CD..
  use IO Completion ports, not memory mapped files
- Network
- Procedural Content Generation
  - Mathematics or Example based
  - Blending Materials
  - Decals / Roads / Vector graphics / Text / Shadows
- Compression? DXT on-load? O(1)?

Examples from Crysis™ *

Terrain material blending (terrain detail objects have been removed

Decals (roads, tire tracks, dirt) used on top of terrain material blending

* not using the virtual texture pixel shader or the combo texture method!

Mip-map generation

- Even Kernel size (e.g. 2x2, 4x4)
  mip-map is offset to the next higher one
  fits to normal GPU behavior
- Odd Kernel size (e.g. 3x3, 5x5)
  mip-map texels are aligned
  good for rectangular chart images?
- Choice affects implementation in other areas
- Non power-of-two: harder, slower, less quality

Out of Core mip-map Generation

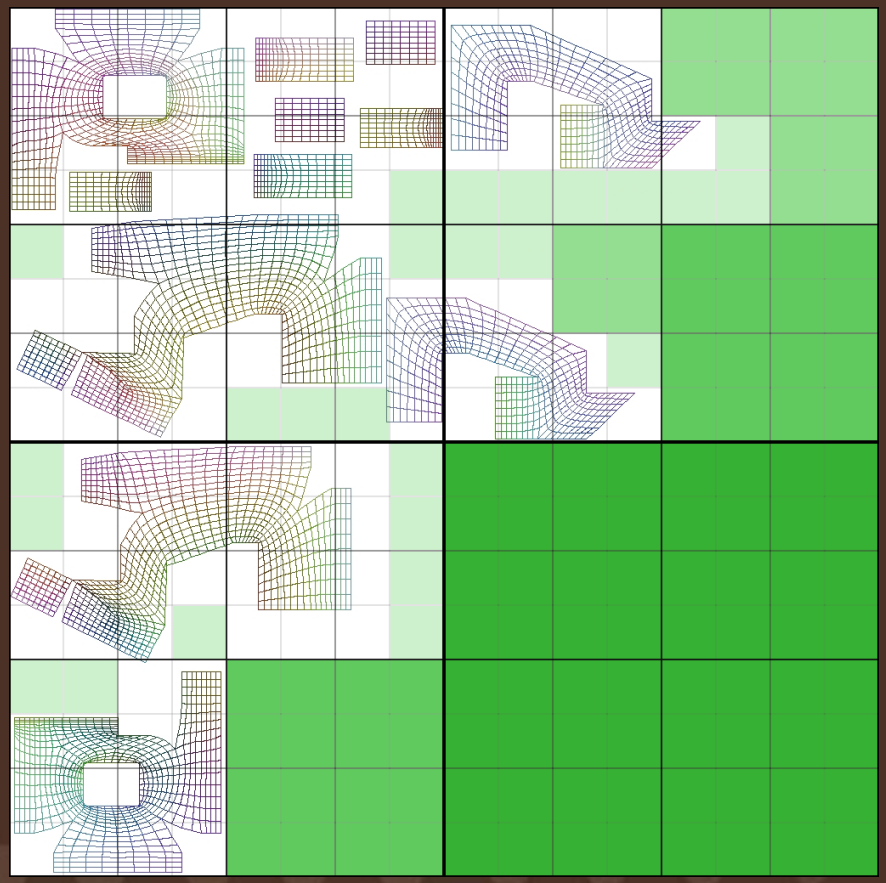
- In memory often not feasible (32Bit)
- Tile based HD layout, no redundancy, uncompressed data, sRGB?
- Functions to request any rectangular block
- Intermediate mip-maps stored in tiles
- Texture compression in export step
- Lazy / on-the-fly mip-map generation

Computing the local LOD

- Conservative or Precise? View-point or View-cone?
- UV mapping dependent?
- min(ddx,ddy) or max(ddx,ddy) or euclidian?
- Many methods e.g.
  - WorldSpace Quad-tree of AABB (tessellation dependent)
  - View space (occlusion problem, single camera)
  - Texture space (GPU, overdraw, resolution?, conservative?)
- D3D9: OcclusionQuery() or CPU readback?

Mesh Parameterization

- Unique unwrapping
  - many applications
  - Workflow issues
  - Stable memory requirements
  - Shading in texture space
- Rectangular charts
- Quad-tree aware unwrapping (less wasted area)
- Sparse textures (e.g. Masks)



Attributes stored in the Tiles

- Similar to the Attributes for Deferred Shading

  Diffuse/Specular Colour, Normal, SpecularPower

- Material ID

  => no blending

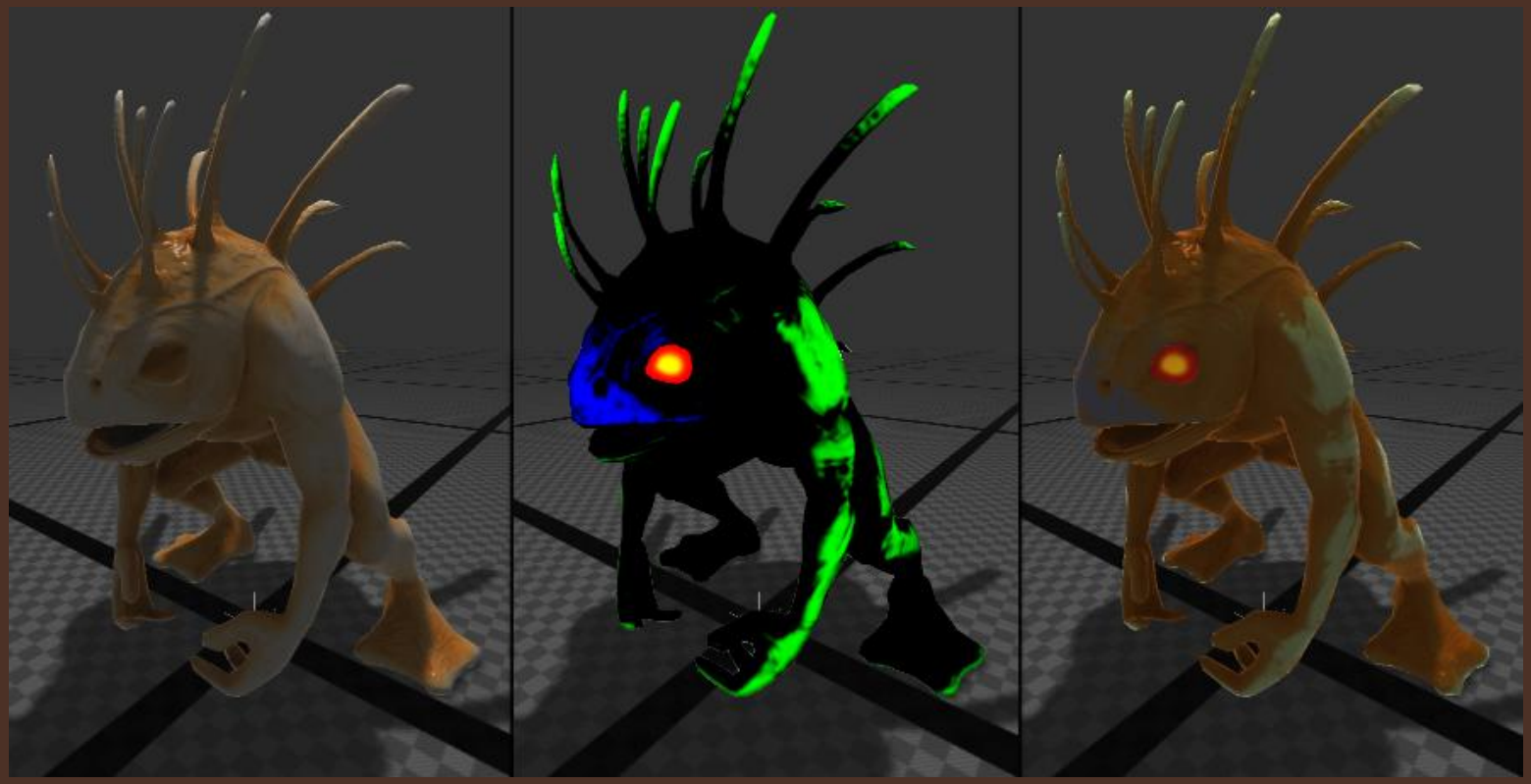
- Material Masks in Sparse Textures

  => blending, compression

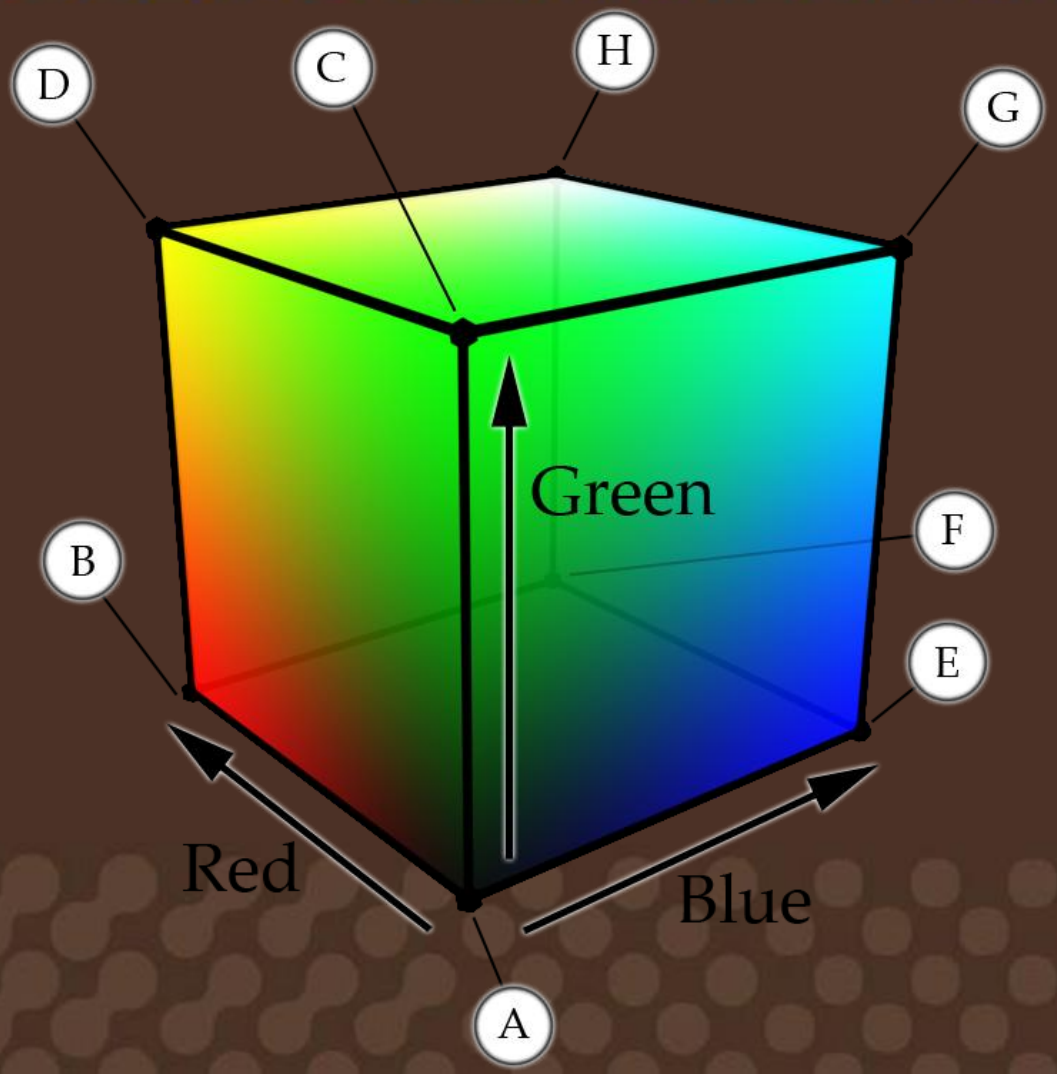
- Combo Texture

  => lossy but static compression

Combo Texture Demo

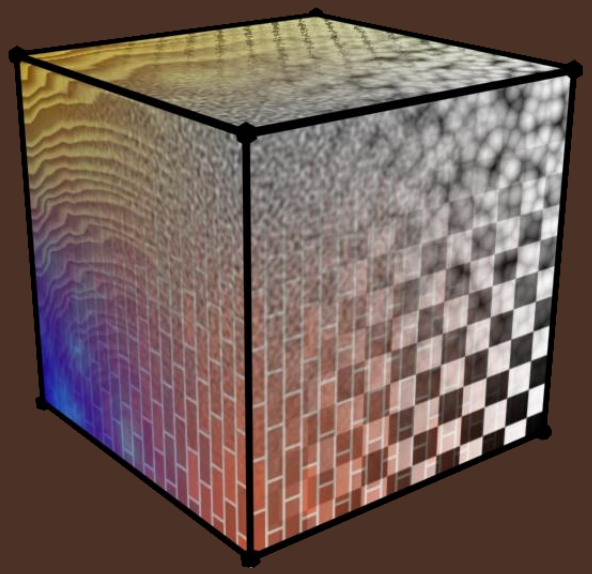


Combo Texture

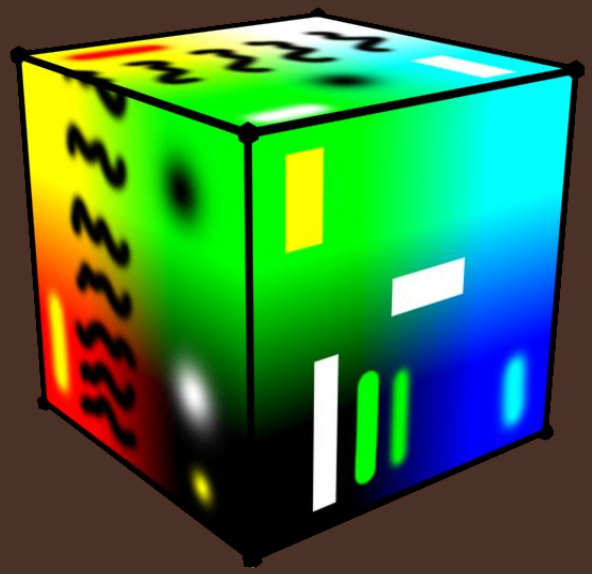


A 3d position in the RGB Cube can be used to blend between up to 8 materials. The blend coefficients can be stored efficiently in a virtual texture

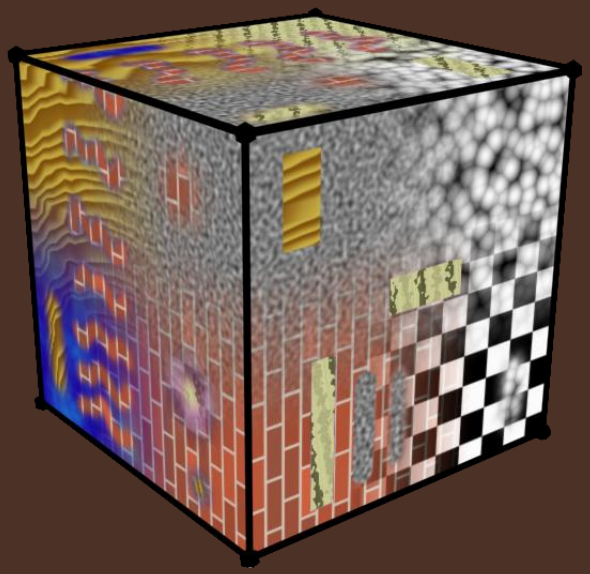
Combo Texture RGB Cube Example



8 materials are defined at the cube corners

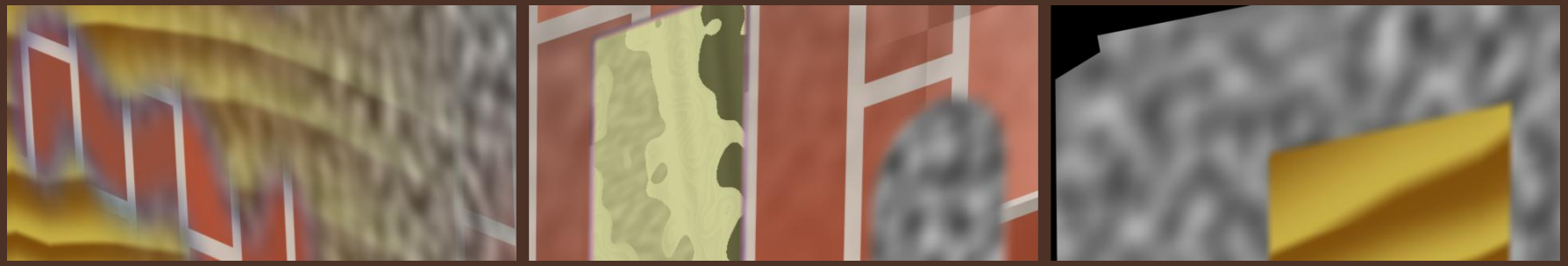


RGB Combo Texture



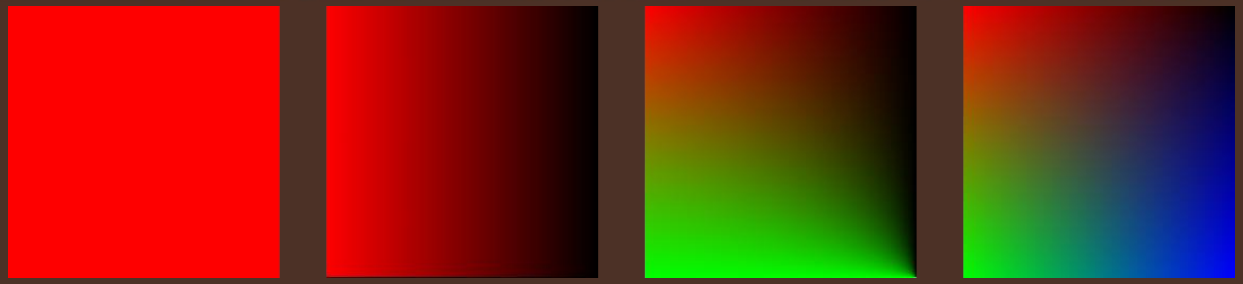
8 materials are blended by the Combo Texture

Combo Texture Properties



- Clean blend only on the cube edges (material placement)
- Bilinear filtering (Custom filtering possible but expensive), DXT
- Dynamic or shader driven combo texture colour (e.g. frost)
- Blending 2/4/8/16 or n materials by using RGB/RGBA texture
  Best: RGB with 2..8 Materials, alpha left for other use
- Single pass only possible in simple cases (e.g. Detail Texture)

Combo Texture Multi pass Blending



4 materials:

One opaque pass
3 passes alpha lerp

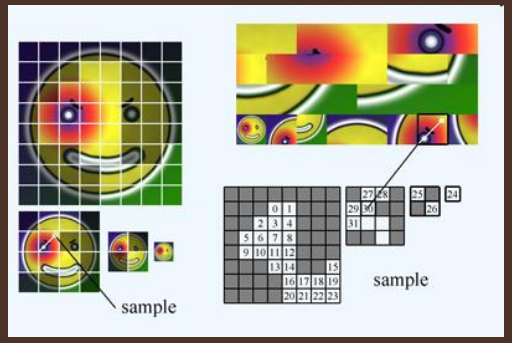
- Alpha needs to be adjusted to compensate the following passes
- Additive blending would be simpler but precision requires FP16
- Less overdraw: Index buffer per material
- Alternative to per triangle material assignment
- Material LOD (e.g. golden buttons on jacket)

Combo Texture Pixel Shader (HLSL)

```
float3 g_ComboMask;                 // RGB material combo colour
                                    // (3 channels for 8 materials)
                                    //   000,100,010,110,001,101,011,111
float4 g_ComboSum0,g_ComboSum1;     // RGBA sum of the masks blended so far
                                    // including the current
                                    // (8 channels for 8 materials)
                                    //   10000000, 11000000, 11100000, 11110000
                                    //   11111000, 11111100, 11111110, 11111111

float ComputeComboAlpha( BETWEENVERTEXANDPIXEL_Unified InOut )
{
  float3 cCombo = tex2D(Sampler_Combo,InOut.vBaseTexPos).rgb;
  float3 fSrcAlpha3 = g_ComboMask*cCombo + (1-g_ComboMask)*(1-cCombo);
  float fSrcAlpha3 = fSrcAlpha3.r*fSrcAlpha3.g*fSrcAlpha3.b;
  float4 vComboRG = float4(1-cCombo.r,cCombo.r,1-cCombo.r,cCombo.r)
                  * float4(1-cCombo.g,1-cCombo.g,cCombo.g,cCombo.g);

  float fSum = dot(vComboRG,g_ComboSum0)*(1-cCombo.b)
             + dot(vComboRG,g_ComboSum1)*(cCombo.b);

  // + small numbers to avoid DivByZero
  return (fSrcAlpha+0.00000001f)/(0.00000001f+fSum);
}
```

Summary

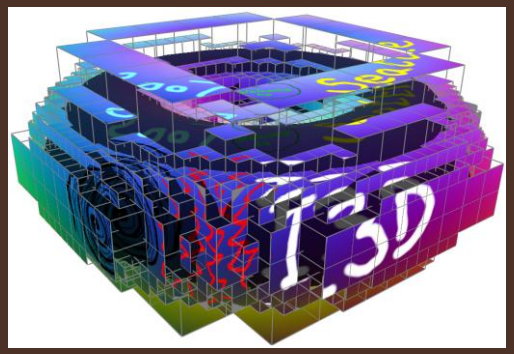
- Many uncovered topics and details
- Virtual Texture is an old idea
- Recommended read:
  - Sparse Virtual Textures
  - Course PDF for details and references

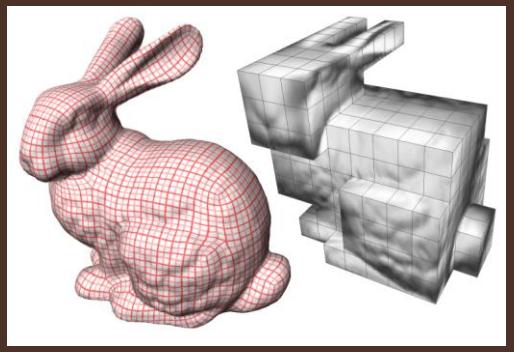


Sparse Virtual Textures
GDC 2008 Sean Barrett

Future

- Better HW / API support
  - Render from/to Virtual Texture
  - Quality of Service (MultiGPU?)
  - Local LOD feedback
  - Compression
- Many Applications
  - Adaptive Shadow Maps
    TileTrees, PolyCube-Maps, …





…


CRYTEK
We are hiring
www.crytek.com/jobs

Acknowledgments

- Efgeni Malachewitsch for the creature 3D model
- Anton Kaplanyan, Nick Kasyan, Michael Kopietz and all others that helped me with this text
- Special thanks to Natasha Tatarchuk, Kev Gee, Miguel Sainz, Yury Uralsky, Henry Morton, Carsten Dachsbacher and the many others from the industry

Questions?