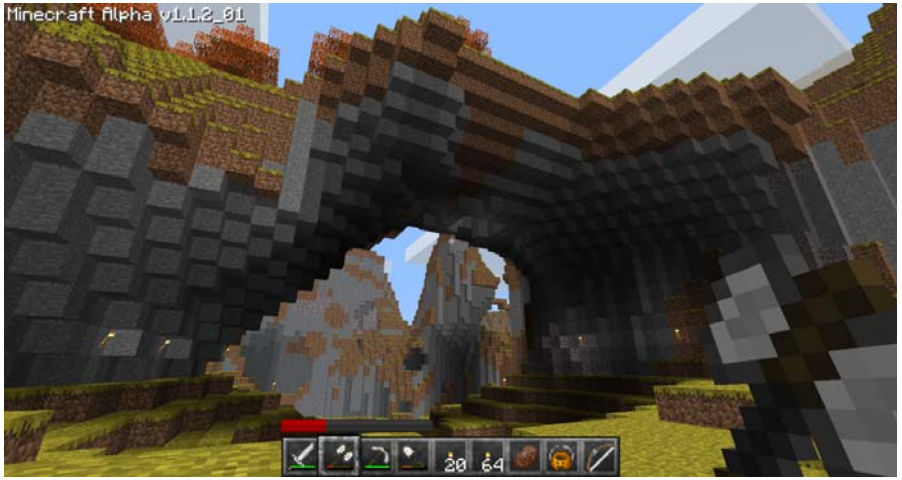
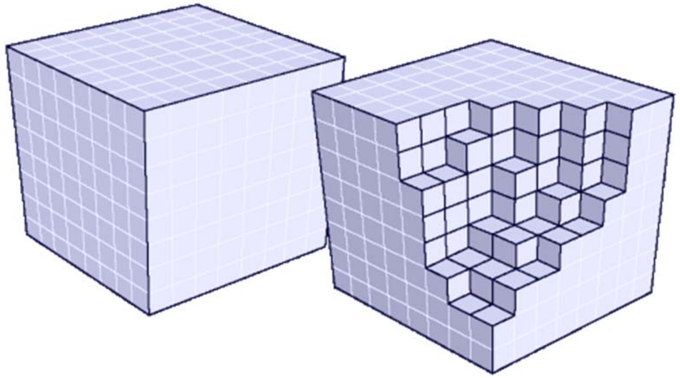
cs7055: Real-time Rendering

Volume Rendering

Interactive
Entertainment
Technology

Voxels





- Analagous to pixels (picture elements), voxels (volume elements) are a discretised representation of 3D space
  - Spatial subdivision of 3D environment
  - Traditionally: environment into homogeneous regular cubes i.e. discrete scalar field
  - Some extensions: object space discretisation, vector/tensor fields

Screenshot of the gameminecraft taken from p://www.nucleocide.net/

Advantages

- Volumetric representation is arguably “real” 3D
  - Physically more accurate e.g. For simulation, physics: destruction, finite elements, fluids
  - More structural information in models
    - Interior details
    - Transparency
    - Fuzzy boundaries
    - Participating media
  - Illumination is not only a function of surface (e.g. Sub surface scattering)
- Potentially more appropriate discretization for rasterization:
  - Voxel to pixel mapping better than triangle to pixel mapping or texel to pixel
  - Can account for effects generated by parallax, displacement, bump-mapping
- Data more uniform – potentially more parallel

Volume Effects

Cloudtank effect in Independence Day

Volume Effects

Nightcrawler's "Bamf" effect in X2

Volume Effects

Digital Avalanche in xXx

Volume Effects

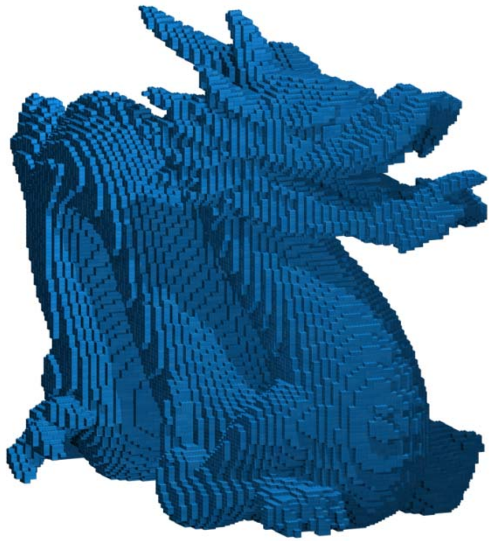
http://http.developer.nvidia.com/GPUGems/gpugems_ch39.html

Particle Approximation

- Some volumetric effects (animation and rendering) can be efficiently approximated by particle systems.
- However:
  - Potential errors in intersection between sprites and scene geometry
  - Accurate lighting shadowing is difficult
  - Animated particle textures can use a lot of memory

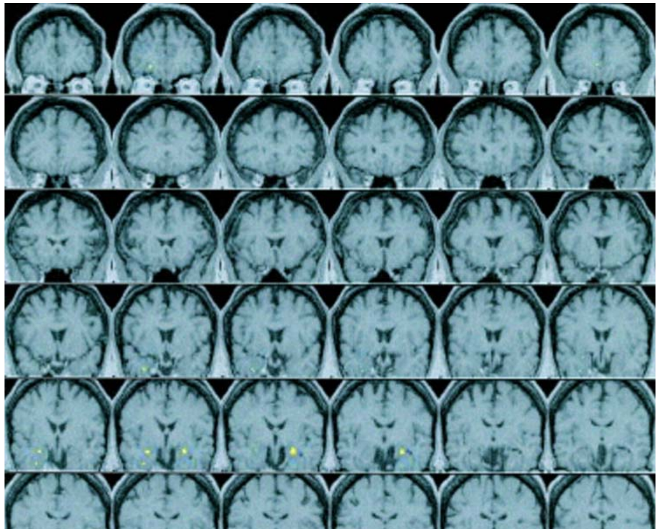
Challenges

- More data (some of it redundant)
  - $256^3 \approx$ 16Mb
  - Large resolutions required to avoid looking blocky
- More complex operations for rendering equation
- Traditional graphics hardware driven more towards accelerating surface & texture models
- Difficult to manually model, edit
- Difficult to understand if not rendered carefully

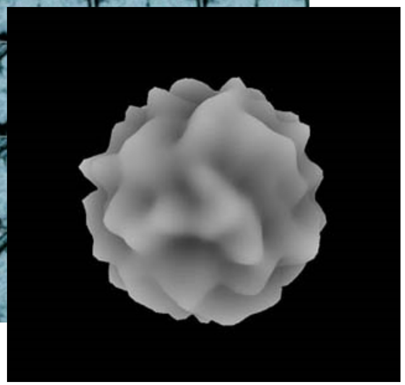


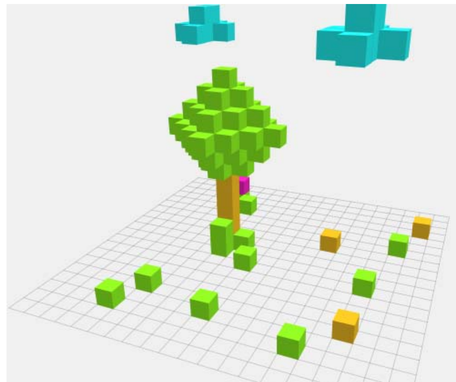
Interactive Entertainment Technology

Volume data



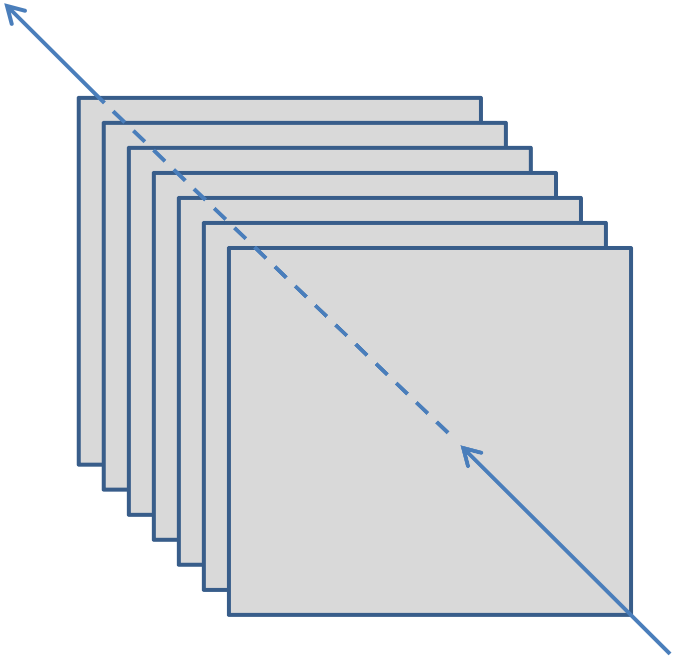
- Sources of volume data:
  - Scanned: e.g. CT, MRI
  - Procedural or simulated
  - Computed from surface: e.g. voxelised (baked)
  - Artist generated: simple volumes e.g. voxel games: minecraft, voxatron



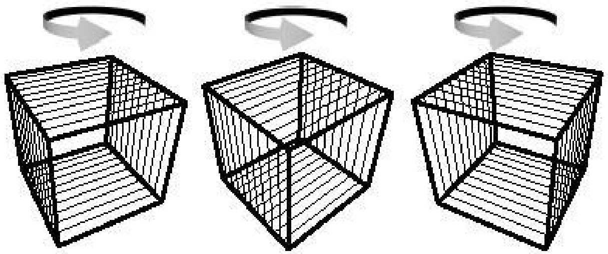


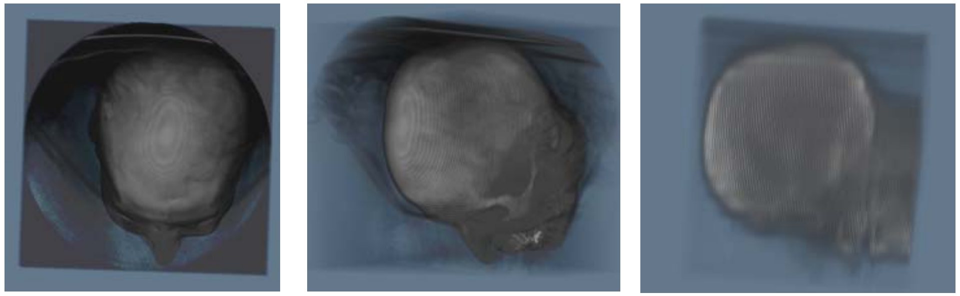
MR dataset: http://jnnp.bmj.com/content/73/6/657.full
Procedural 3d noise function: http://developer.nvidia.com/object/cg_effects_explained.html
Voxelizer: http://www.luima.com/voxpro.htm
Online voxel editor: http://mrdoob.com/projects/voxels/

Slice Rendering



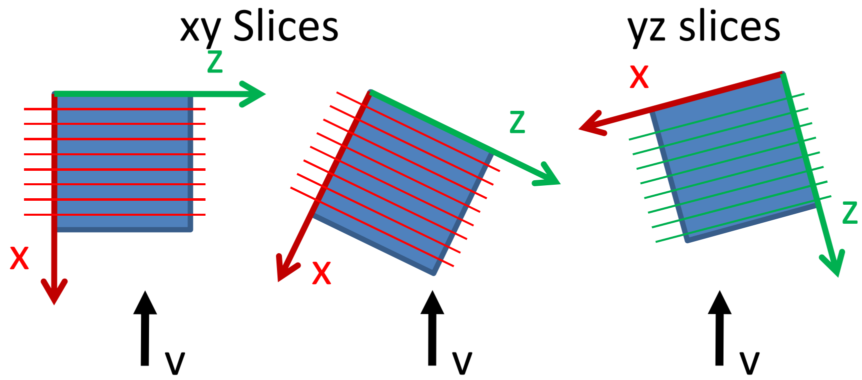
- Slice blending
  - Simple render back to front with alpha blending
- Problems:
  - opacity dependent on orientation
  - when view is rotated away from “scan direction” we start to see gaps between slices



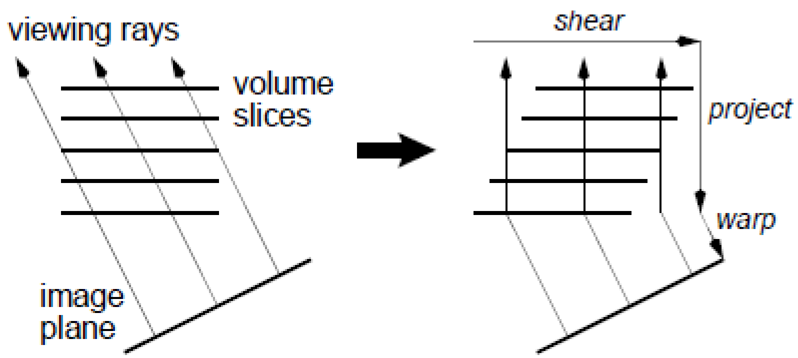


View Dependent Slices

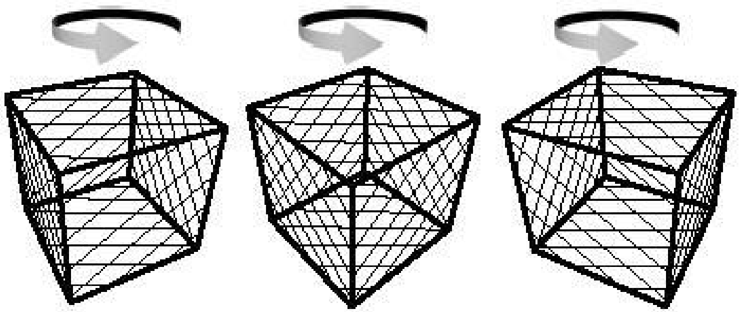
- Solution:
  - Convert slices to 3D voxel array
  - “Slice” along different axis directions based on closest axis to view direction



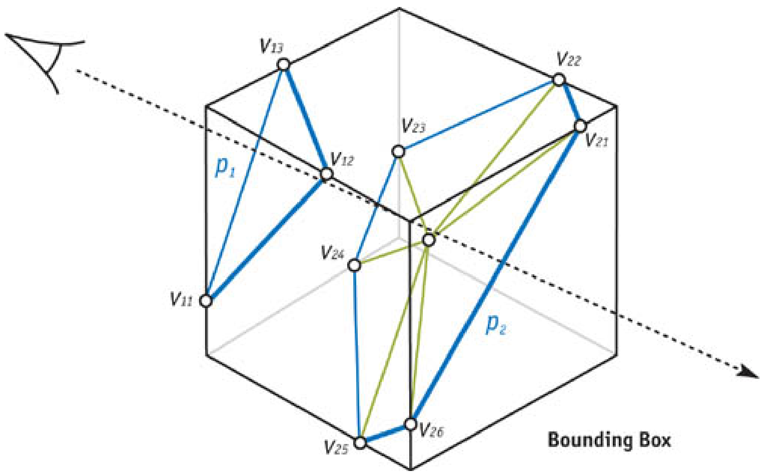
- Shear Warp technique: to fix opacity variation at intermediate viewing angles
  - Slices are sheared (progressively moved towards centre of view)
  - Slices are warped to account for viewing distortion



View Aligned Slices



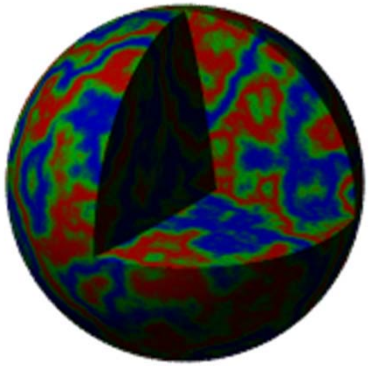
Viewport-Aligned Slices



1. Transform volume bounding box vertices using the modelview matrix.
2. Compute view orthogonal sampling planes, based on:
   - ◇ Distance between min and max *z* of bounding box verts
   - ◇ Equidistant spacing scaled by voxel size and sampling rate.
3. For each plane
   a) Test for intersections with bounding box. Generate a **proxy polygon** (upto 6 sides).
   b) Tessellate proxy polygon into triangles and add the resulting vertices to the output vertex array
   c) Generate texture coordinates for each triangle vertex

Volumetric Textures

- Texture mapping may be applied not only on the surface
- Volumetric textures define mapping 3D
  - Mostly procedural
    - Generators e.g. turbulence hlsl, noise glsl
  - More commonly used as textures in off-line renderings
  - For real-time, hardware support available. Several hardware related advantages:
    - direct 3D addressing
    - tri-linear interpolation
    - 3D coherent texture caching
  - N.B. Memory limitations!
    - 512^3 3D texture with 1 byte values takes over 128MB



Volume Ray Casting

- **Ray casting.** For each pixel of the final image, cast eye ray through the volume (usually enclosed within a *bounding box* used to intersect the ray and volume).
- **Point Sampling.** equidistant *sampling points* or *samples* are selected along ray. Sampling points usually will be located in between voxels so tri-linearly interpolate values from surrounding voxels.
- **Point Shading.** For each sampling point either:
  - Apply some colour based on sampled value and a transfer function: classical Direct Volume Rendering (DVR)

  OR
  - Calculate gradient (orientation of local surfaces) and calculate illumination
- **Compositing.** Combine shaded samples to get the final colour value for the ray.

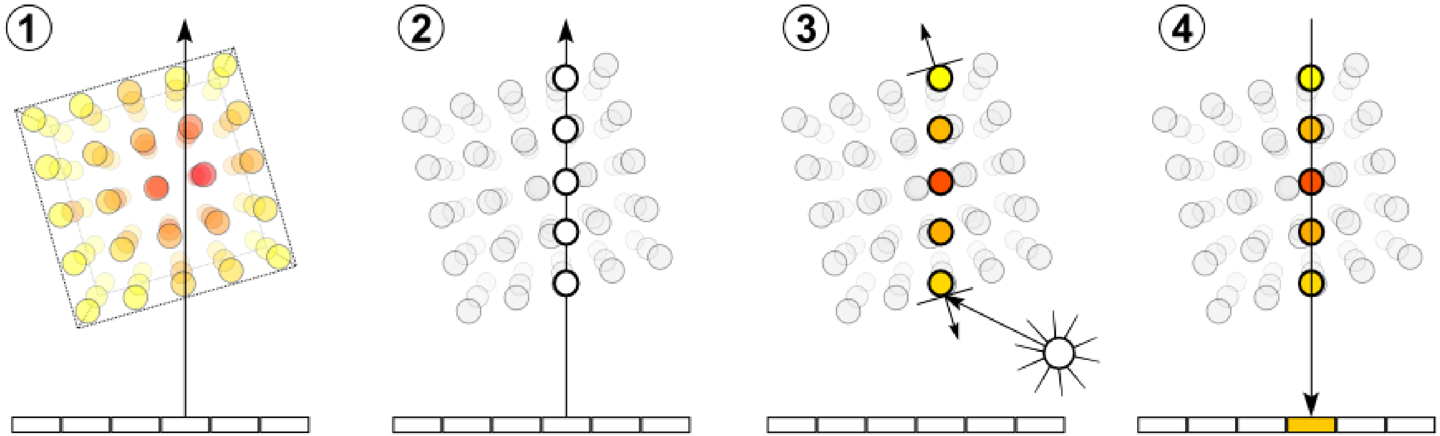
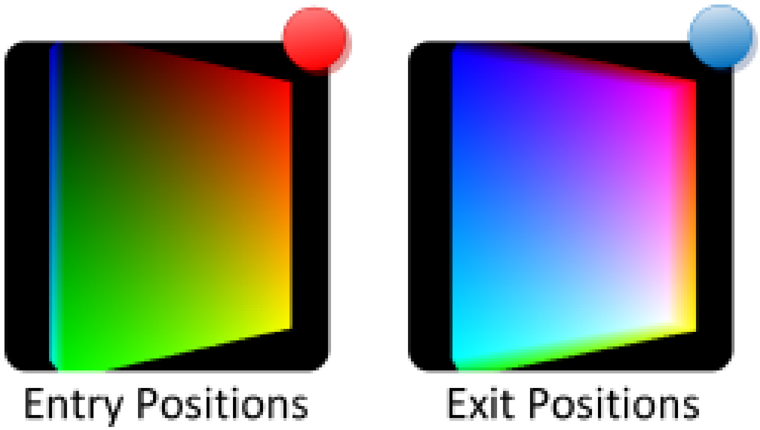
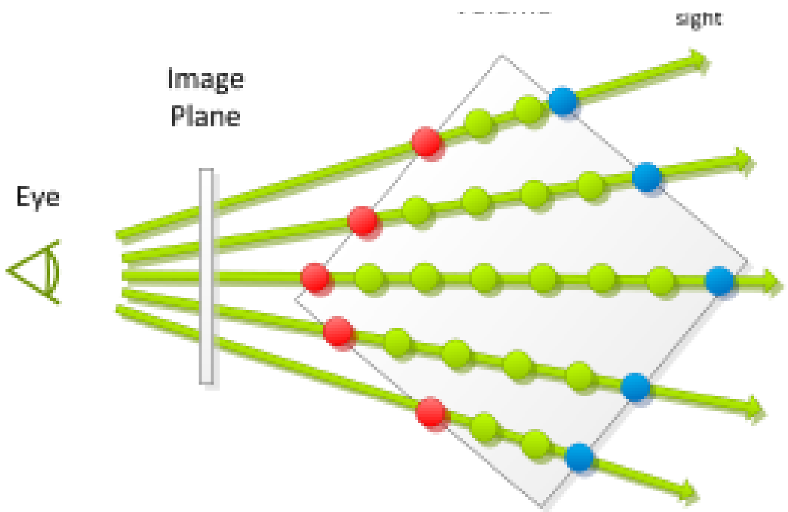


Image from wikipedia: http://en.wikipedia.org/wiki/Volume_ray_casting

GPU Ray Marching

1. Compute volume Entry Position
2. Compute ray of sight direction
3. While in Volume
   A. Lookup data value at ray position
   B. Accumulate Colour and Opacity
   C. Advance along ray





S. Stegmaier, M. Strengert, T. Klein, and T. Ertl. “A simple and flexible volume rendering framework for graphics-hardware-based raycasting”. In Volume Graphics 2005

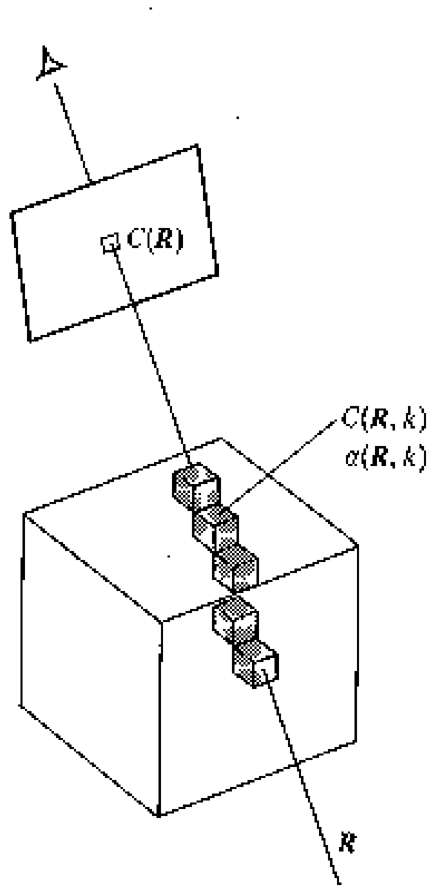
GPU Ray Marching

```
float4 RayMarchPS(Ray eyeray : TEXCOORD0,  uniform int steps) : COLOR
{
        eyeray.d = normalize(eyeray.d ); d

        // calculate ray intersection with bounding box
        float tnear, tnear;
        bool hit = IntersectBox(eyeray , boxMin , boxMax , tnear , tfar );
        if (!hit) discard;
        if ( tnear < 0.0) tnear = 0.0;

        // calculate intersection points
        float3 Pnear = eyeray.o + eyeray.d * tnear;
        float3 Pfar = eyeray.o + eyeray.d * tfar;

        // march along ray, accumulating color
        half4 c = 0;
        half3 step = ( Pnear Pfar ) / (steps 1);
        half3 P = Pfar;
        for (int i=0; i<steps; i++)
         {
              half4 s = VOLUMEFUNC(P);
              c = s.a*s + (1.0 s.a)*c;
              P += step;
        }
        c /= steps;
        return c;
}
```



Ray Marching: As we are intersecting rays with aligned cubes, it is a bit easier than generic intersection testing: rasterization techniques e.g. DDA can be employed

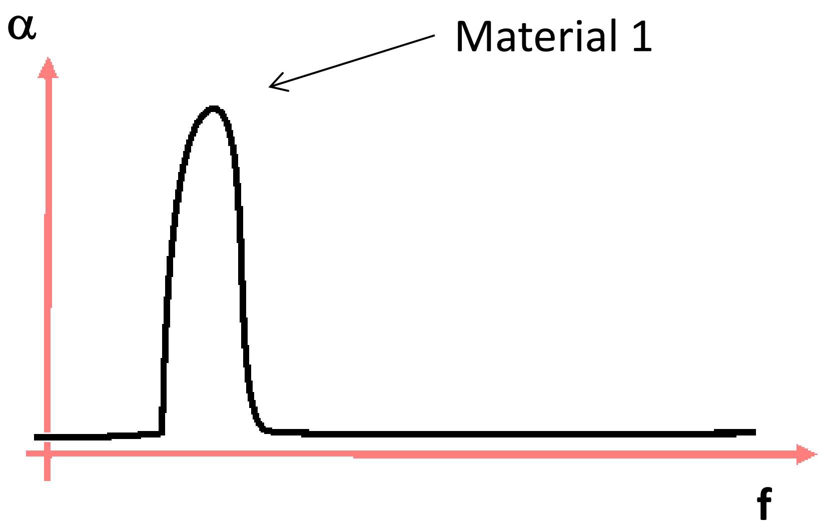
HLSL Code sample from: http://developer.nvidia.com/object/gdc_2005_presentations.html

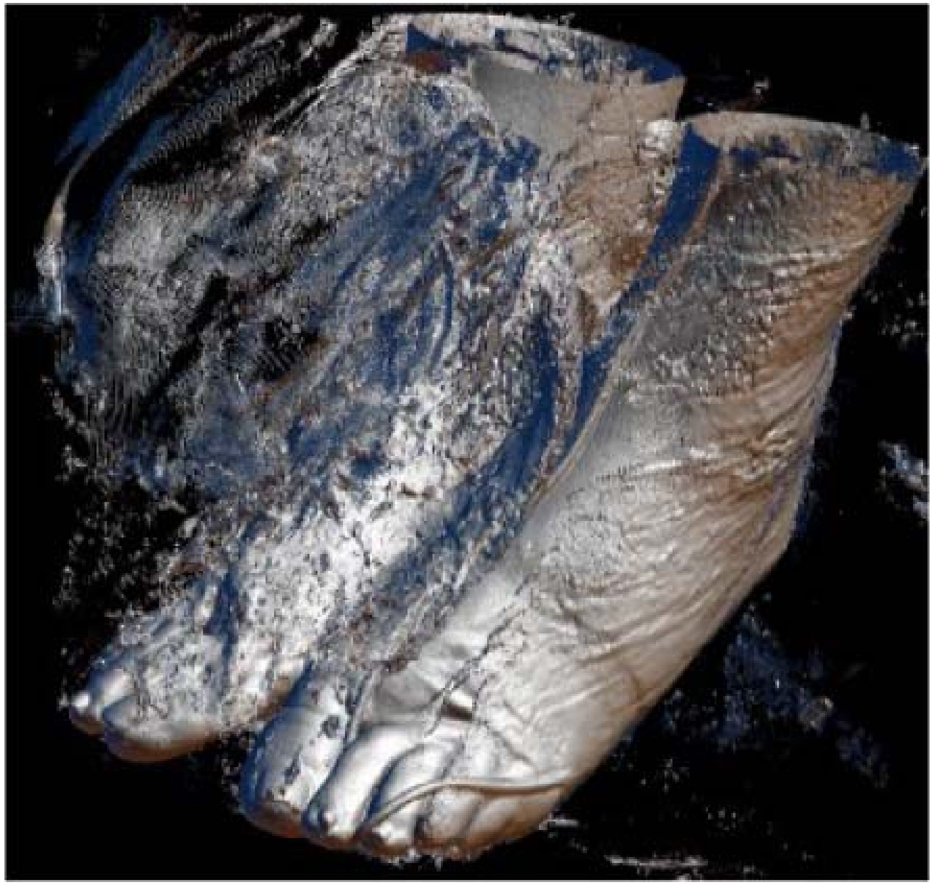
Transfer Function

- Transparent scalar field is difficult to understand (information overload)
- Map scalar values to colours or opacity
  - Interpretive rendering
  - Allows user to choose which levels are more visible OR attach colors/alpha to specific voxel levels
  - Visualisation: make visiual data easier to understand
  - Less important for games

Images from presentation by J. Tierney, Univ. Utah http://www.sci.utah.edu/~jtierny/stuff/teaching/tierny_intro_vol_rend09.pdf

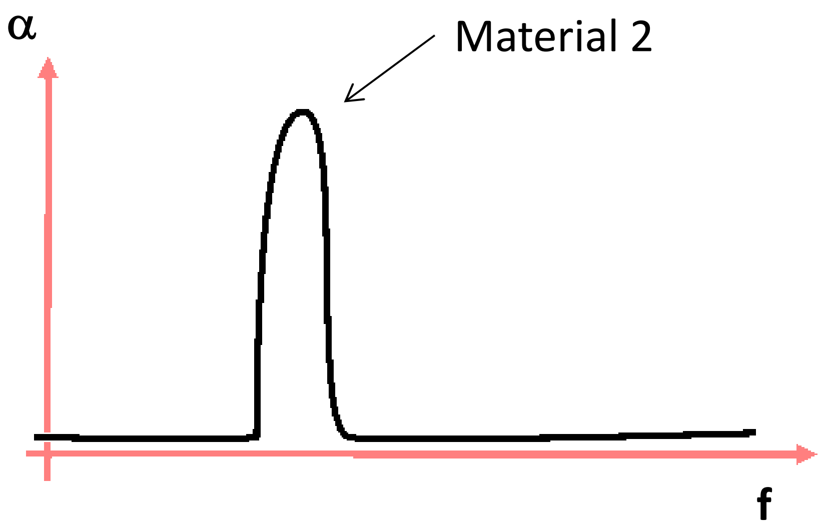
Transfer Function





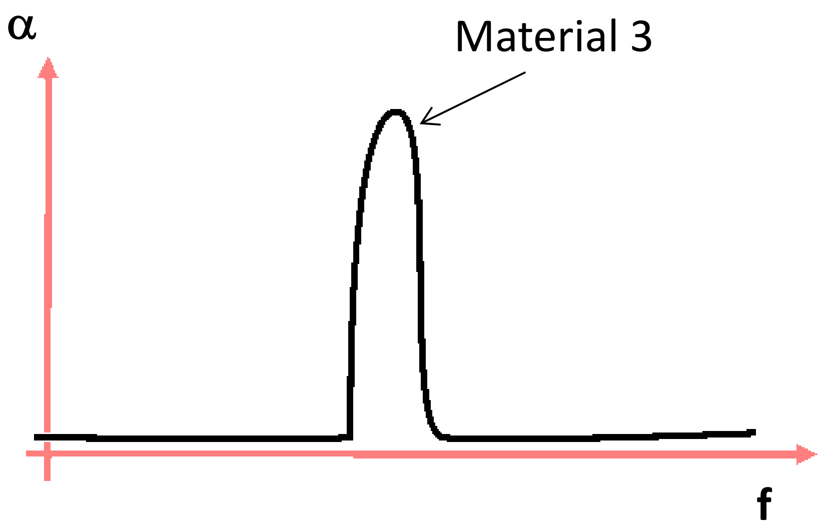
Images from presentation by J. Tierney, Univ. Utah http://www.sci.utah.edu/~jtierny/stuff/teaching/tierny_intro_vol_rend09.pdf

Transfer Function



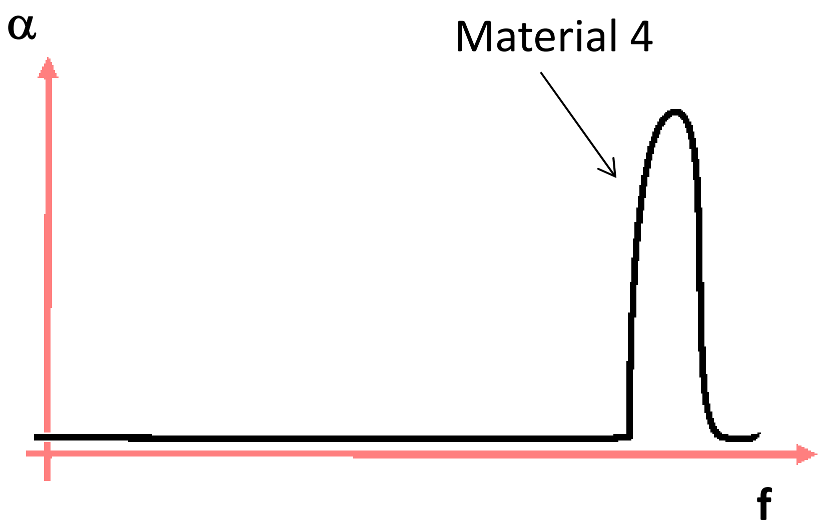
Images from presentation by J. Tierney, Univ. Utah http://www.sci.utah.edu/~jtierny/stuff/teaching/tierny_intro_vol_rend09.pdf

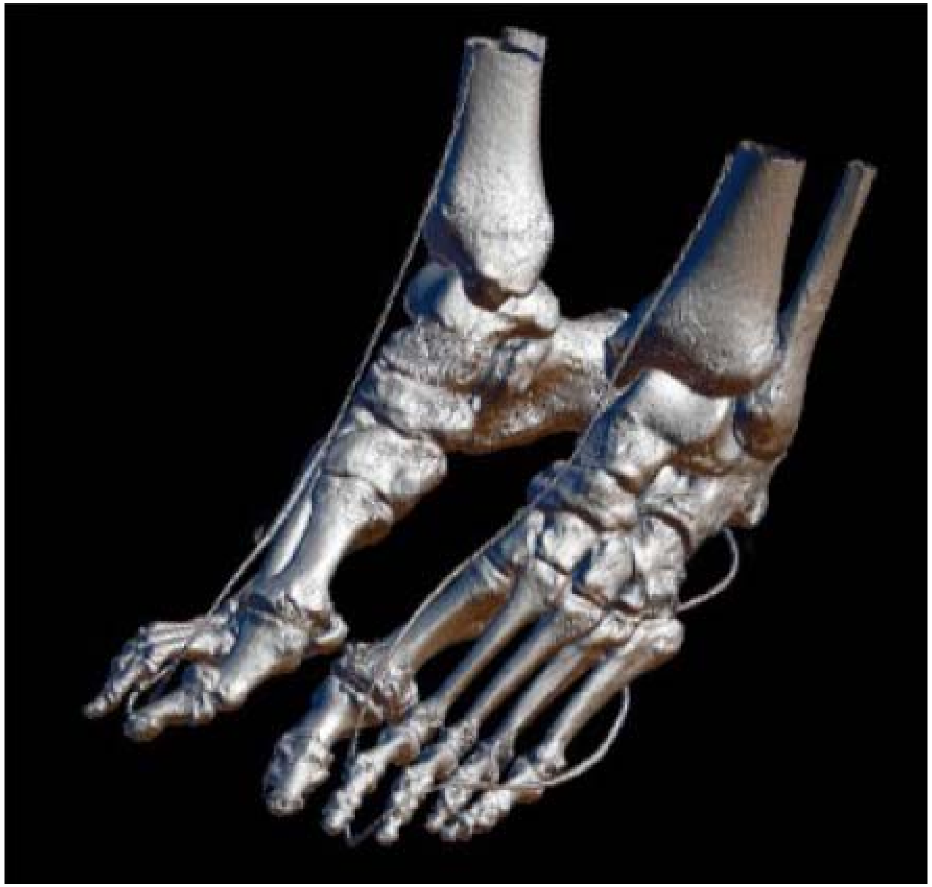
Transfer Function



Images from presentation by J. Tierney, Univ. Utah http://www.sci.utah.edu/~jtierny/stuff/teaching/tierny_intro_vol_rend09.pdf

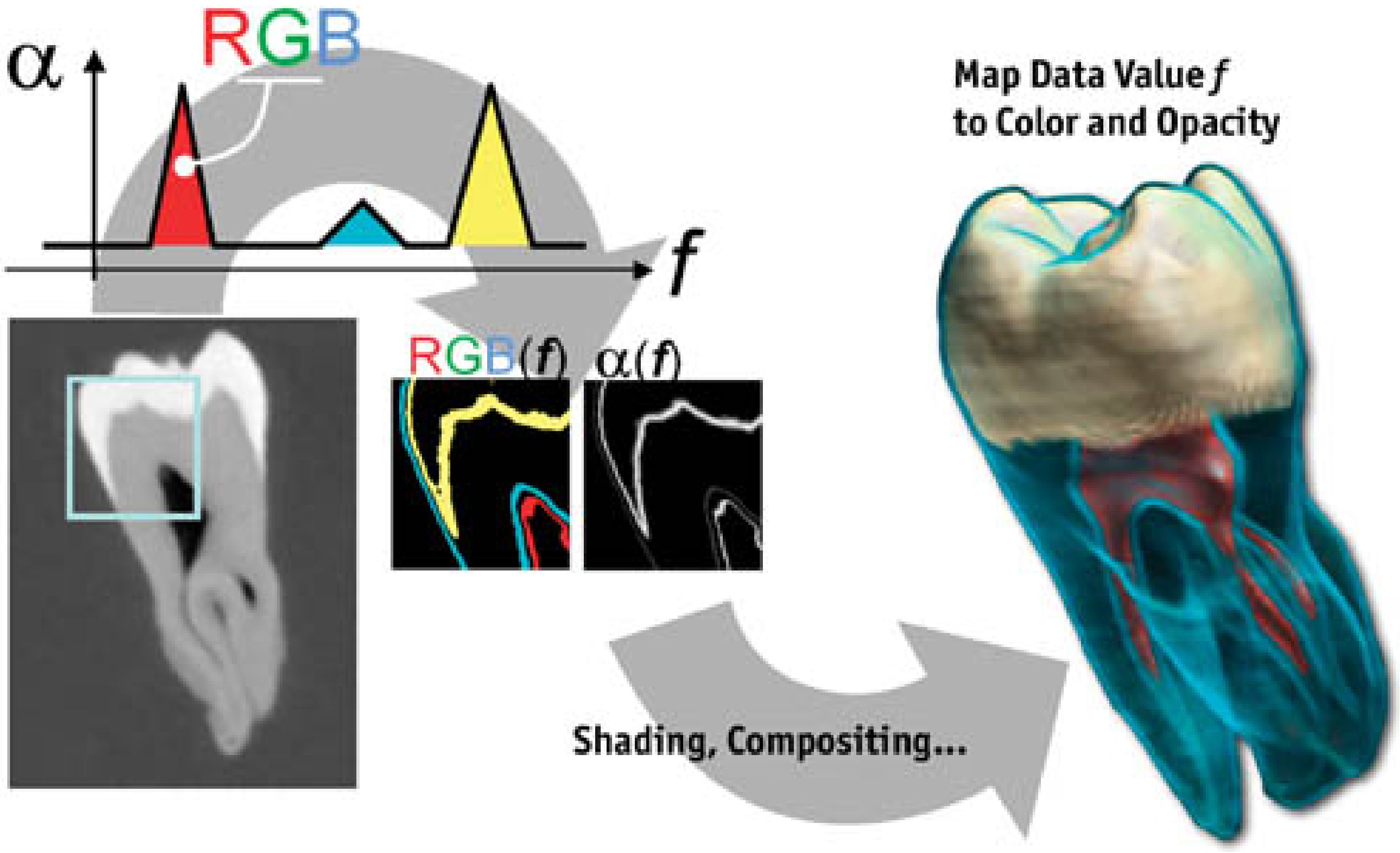
Transfer Function





Images from presentation by J. Tierney, Univ. Utah http://www.sci.utah.edu/~jtierny/stuff/teaching/tierny_intro_vol_rend09.pdf

Transfer Function



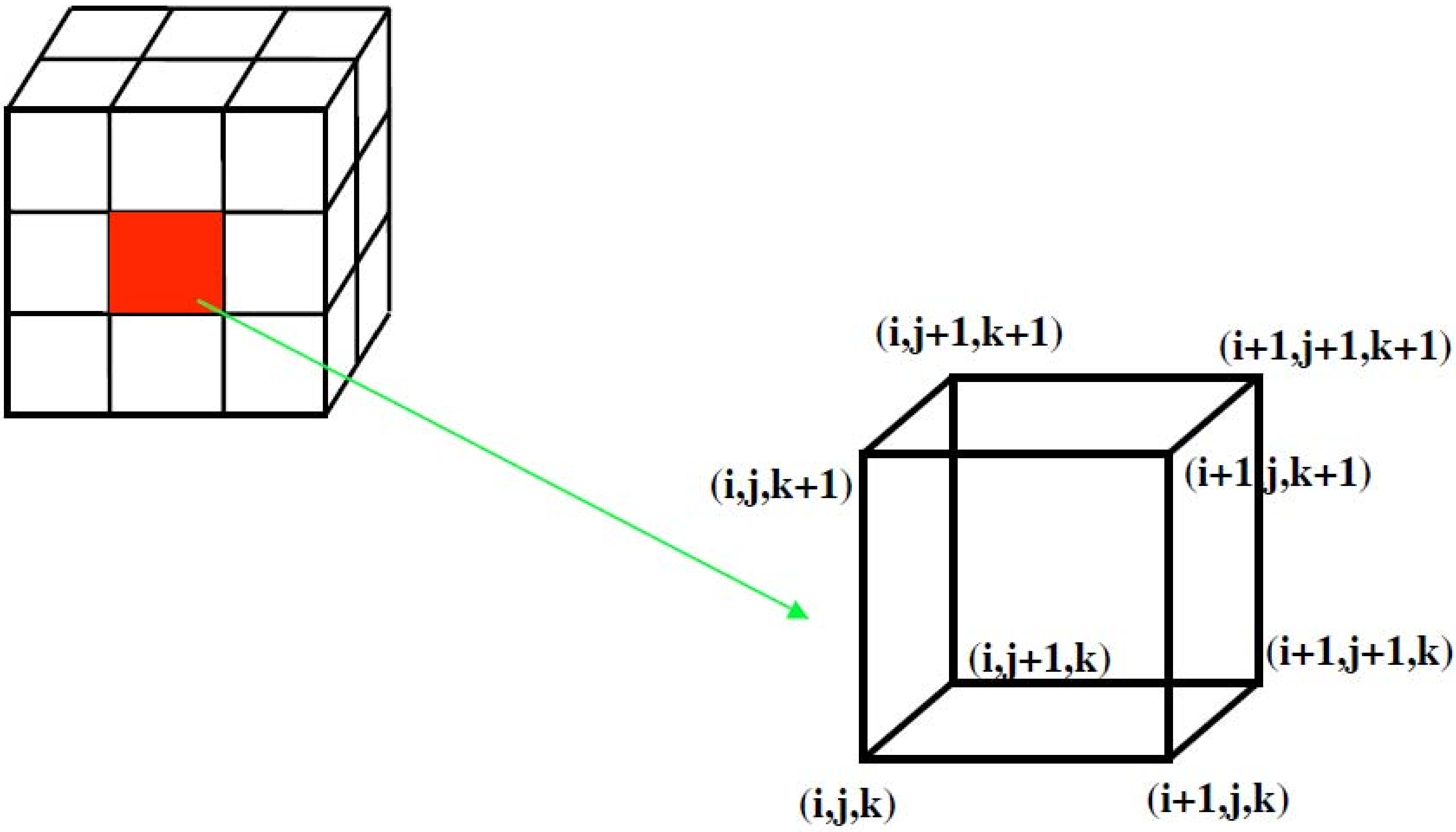
http://http.developer.nvidia.com/GPUGems/gpugems_ch39.html

Interactive Entertainment Technology

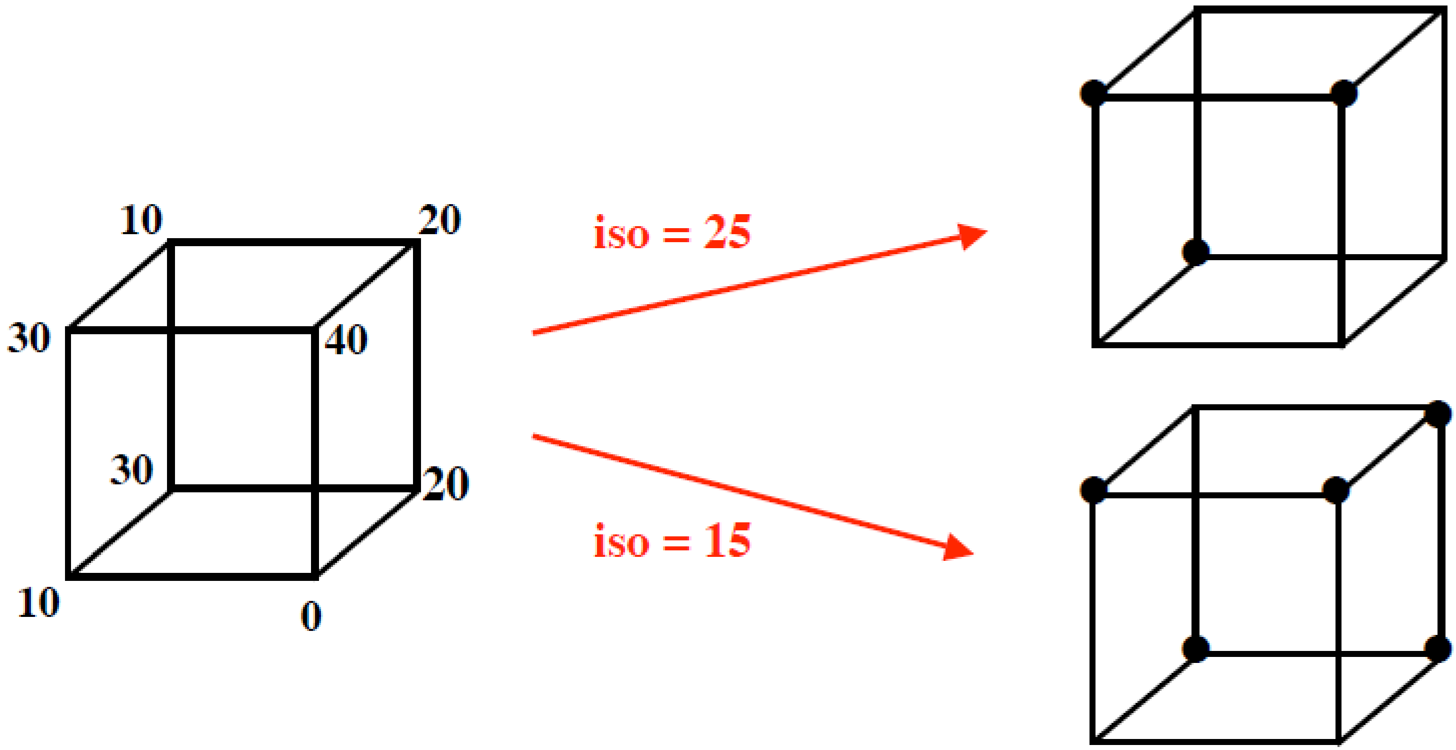
Indirect Volume Rendering

- There are some benefits to surface based techniques when it comes to rendering
  - More traditional pipeline optimizatios
  - Accurate reflections
  - Clear boundary representation
- Indirect Volume Rendering techniques first extract one or more iso-surfaces from the volume data
  - Alternatively render one isosurface and blend it with DVR
- Either:
  - Implicitly/on-the fly
  - Iso-surface mesh extraction

Marching Cubes

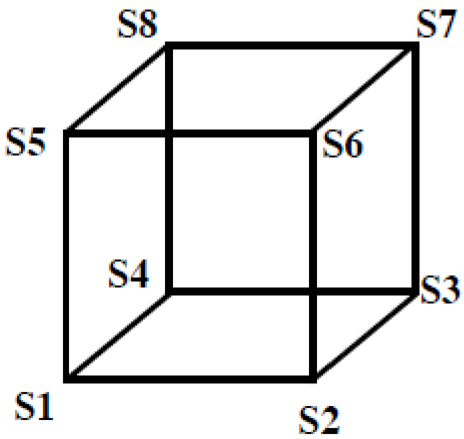


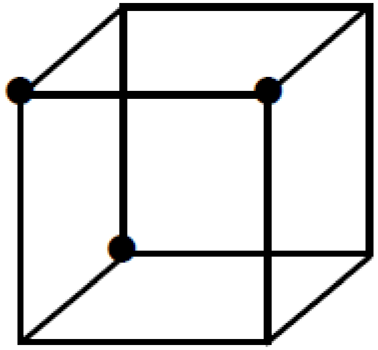
Marching Cubes



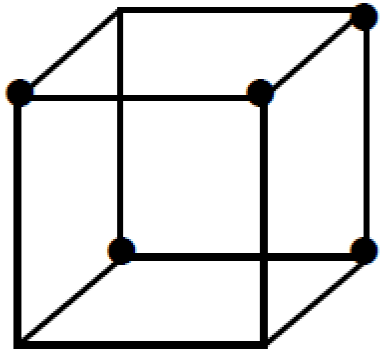
Marked vertex by ● = inside = 1

Unmarked vertex = outside = 0





00011100



?

index	S1	S2	S3	S4	S5	S6	S7	S8

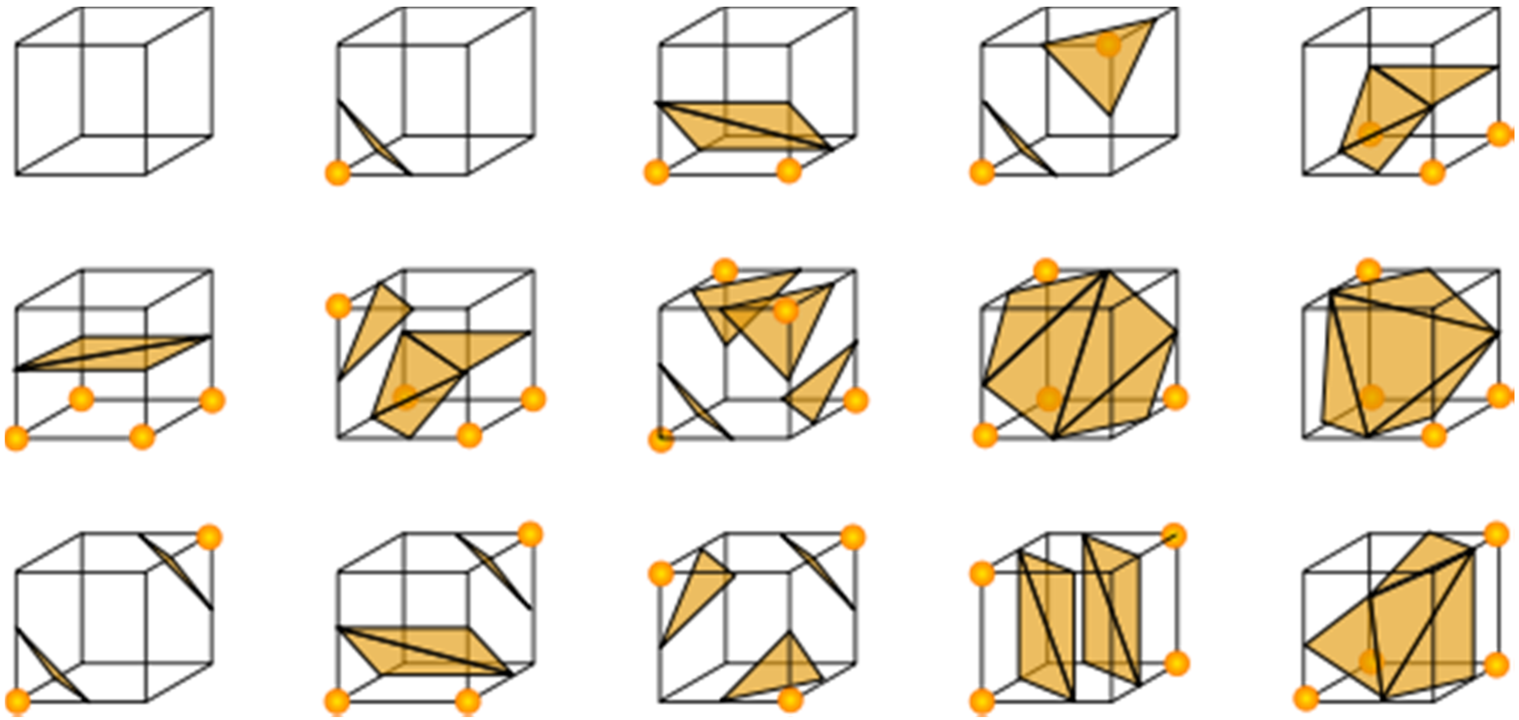
or

index	S8	S7	S6	S5	S4	S3	S2	S1

Forms the bits of a binary number between 0 and 255 for an 8-vertex cube

Marching Cubes

- Removing redundant cases e.g. completementary and rotational symmetries: each voxel is identified as one of 15 cases:

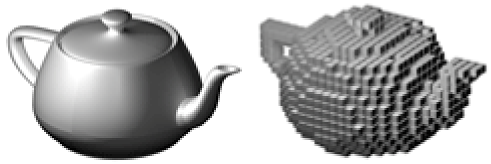


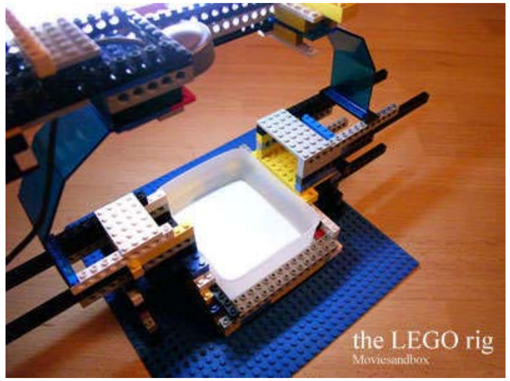
optimize with e.g. Octree

Voxelization

- Conversely, we sometimes want to extract a volume representation of surface models
- Basic algorithm:

  1. Find bounding box for object: get wx, wy, wz
  2. Choose slice resolution e.g. slice_depth = min (wx, wy)
  3. Let z = wz
  4. Choose a clipping plane orthogonal to z_axis distance z from near clipping plane viewer
  5. Render object to texture (texture_slice[z])
  6. Let z = z - slice_depth
  7. Repeat from 4 until z = 0
  8. Repeat from z=0 and increasing values of z (AND values from two sweeps)
  9. For non-convex objects we must also do, x, -x, y, -x directions (form of space carving)





- Some problems:
  - doesn’t handle some holes and concavities
  - Boolean values for voxels. Doesn’t do range (surface models not great for this anyway)

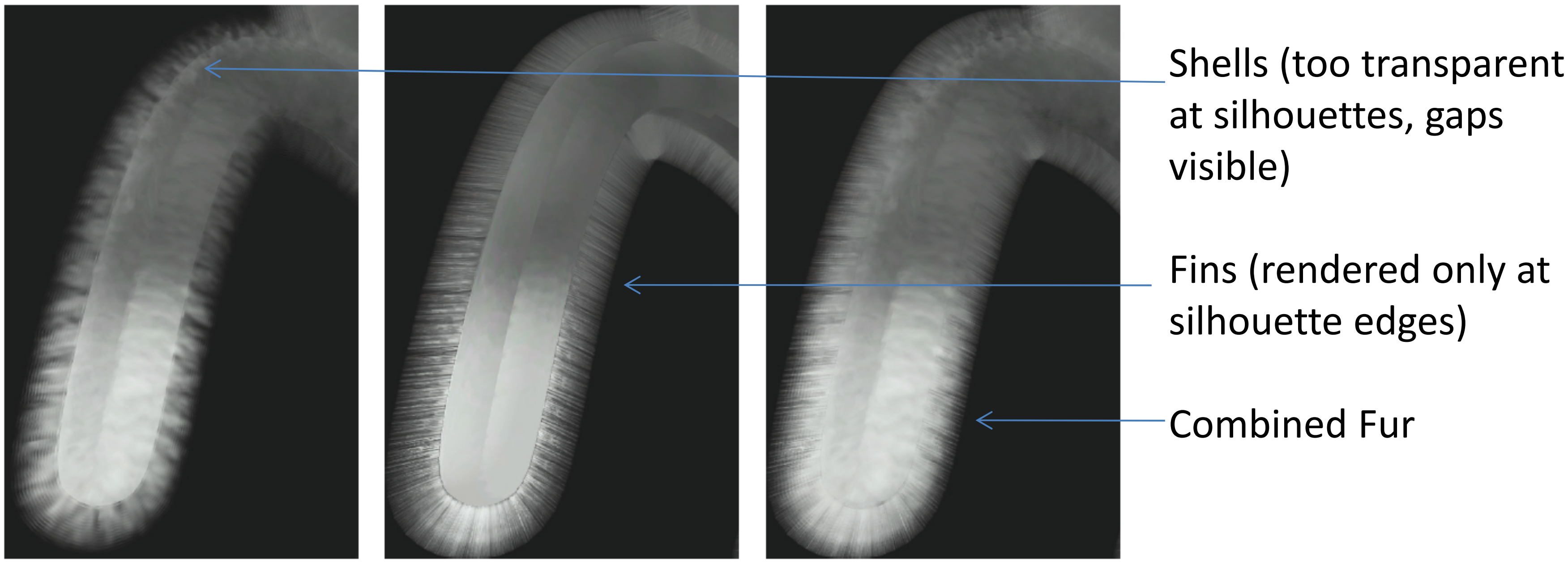
For a low-tech method of voxelizing real world objects see the video of the milk scanner:
http://www.youtube.com/watch?v=XSrW-wAWZe4

Interactive Entertainment Technology

Applications to Games

Surface Volumetric Textures

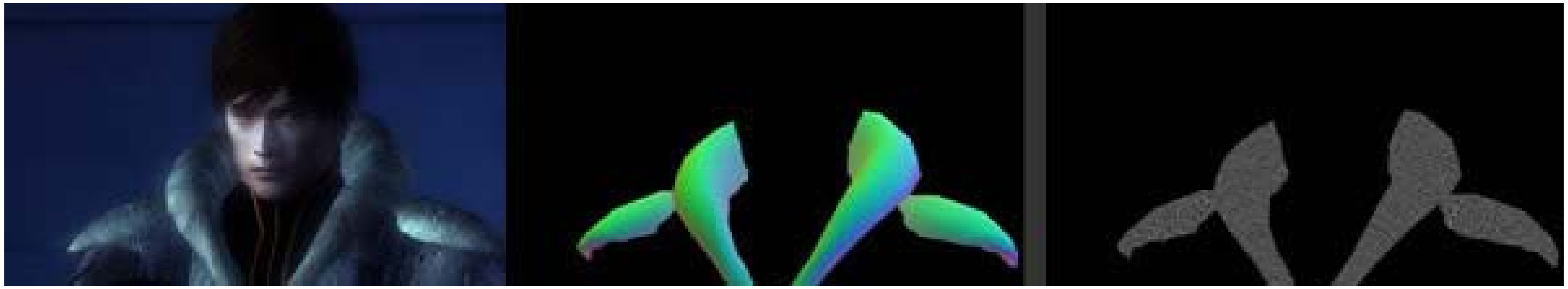
- Layers of 2D textures – but usually biased around the surface of an object (more tied into traditional pipeline)
- Good for complex surfaces: landscape details, organic tissues, fuzzy/hairy objects
- Nested shells: surface texture is rendered multiple times, extruding a little each time. Fins are extra extruded geometry rendered to improve silhouette



http://developer.download.nvidia.com/whitepapers/2007/SDK10/FurShellsAndFins.pdf

Surface Volumetric Textures

Geometry shaders are used in Lost Planet to extrude textures outwards based on normal map



Colour Normal Length

Lost planet Images from http://meshula.net/wordpress/?p=124

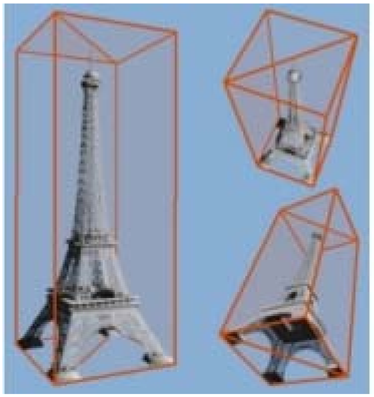
Fur rendering in Furmark

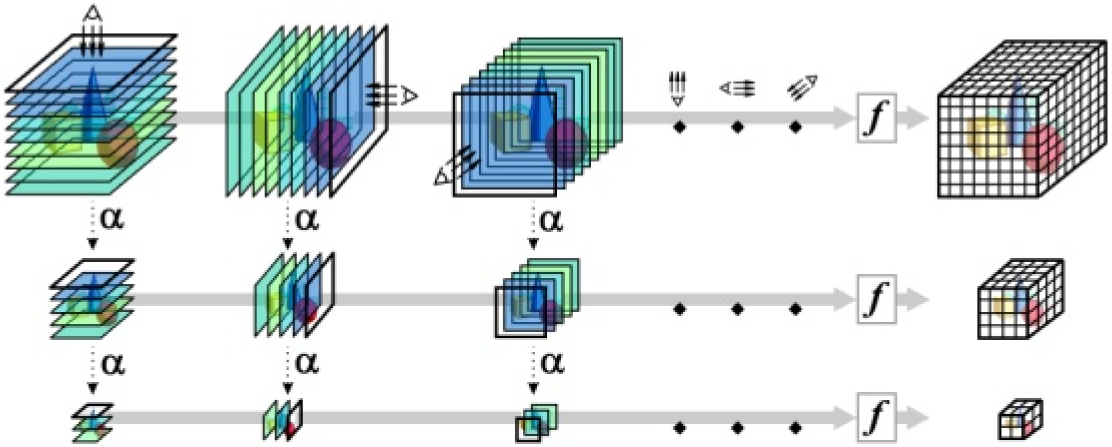
3D Textures for Fur

N.B. 3D texture for fur is not new. Above off-line rendered images by Kajiya in 1989 but rendering time was 2 hours!

Volumetric Billboards

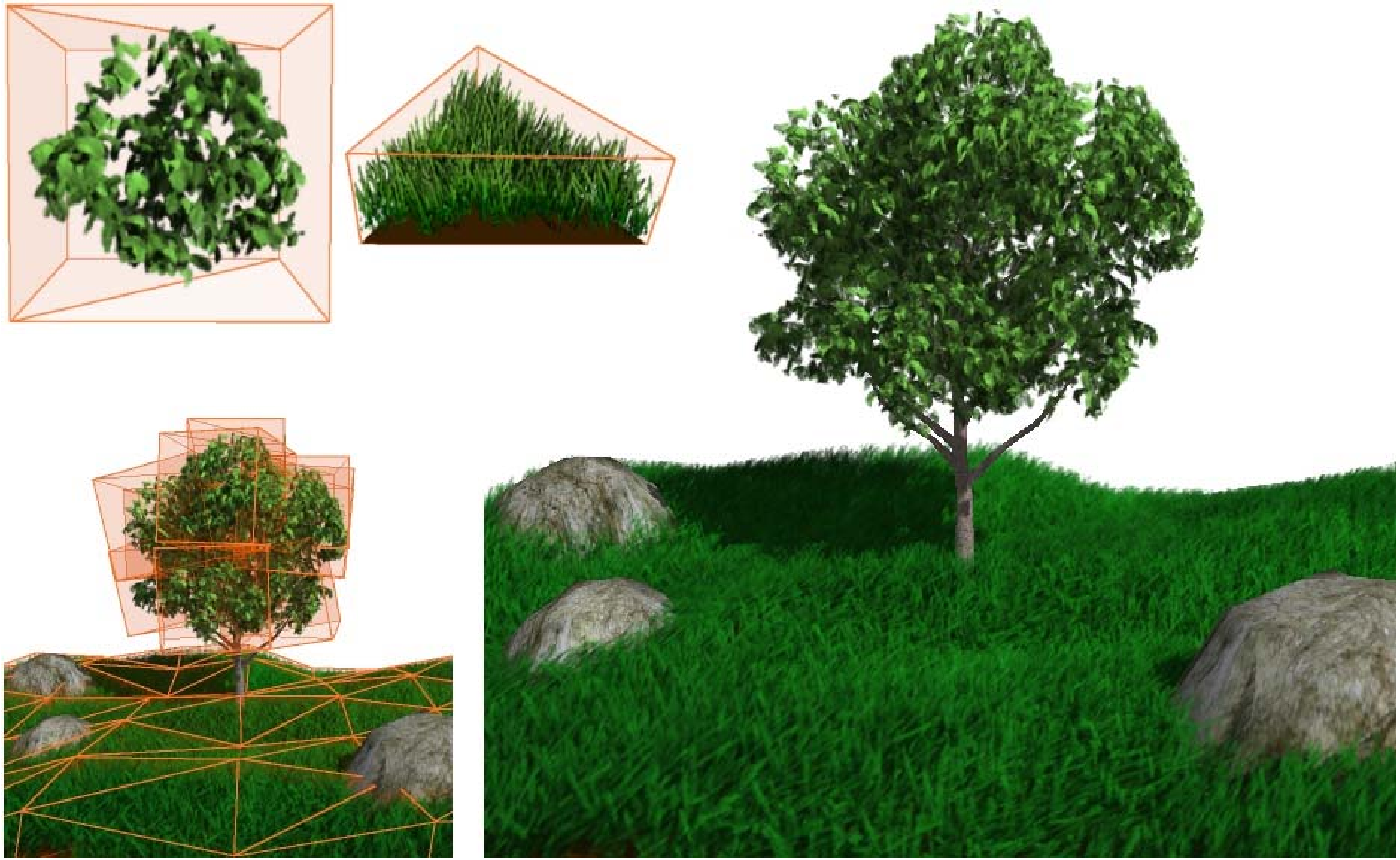
- Uses 3D textures instead of traditional 2D for billboards
  - Exploit Geometry Shader for real-time
  - Full-parallax effect – without popping artifacts
  - Combine with mip-mapping for Level of Detail





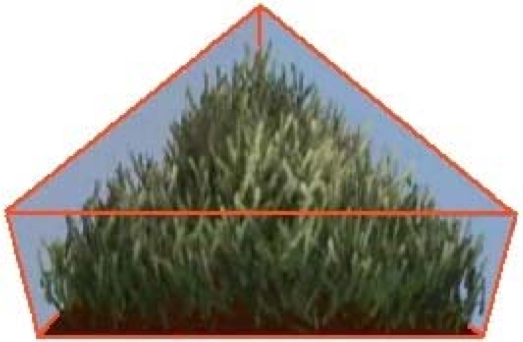
Decaudin and Neyret "Volumetric Billboards" http://www.antisphere.com/Research/VolumetricBillboards.php

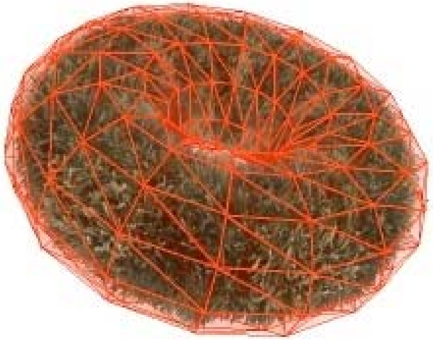
Volumetric Billboards



Decaudin and Neyret “Volumetric Billboards” http://www.antisphere.com/Research/VolumetricBillboards.php

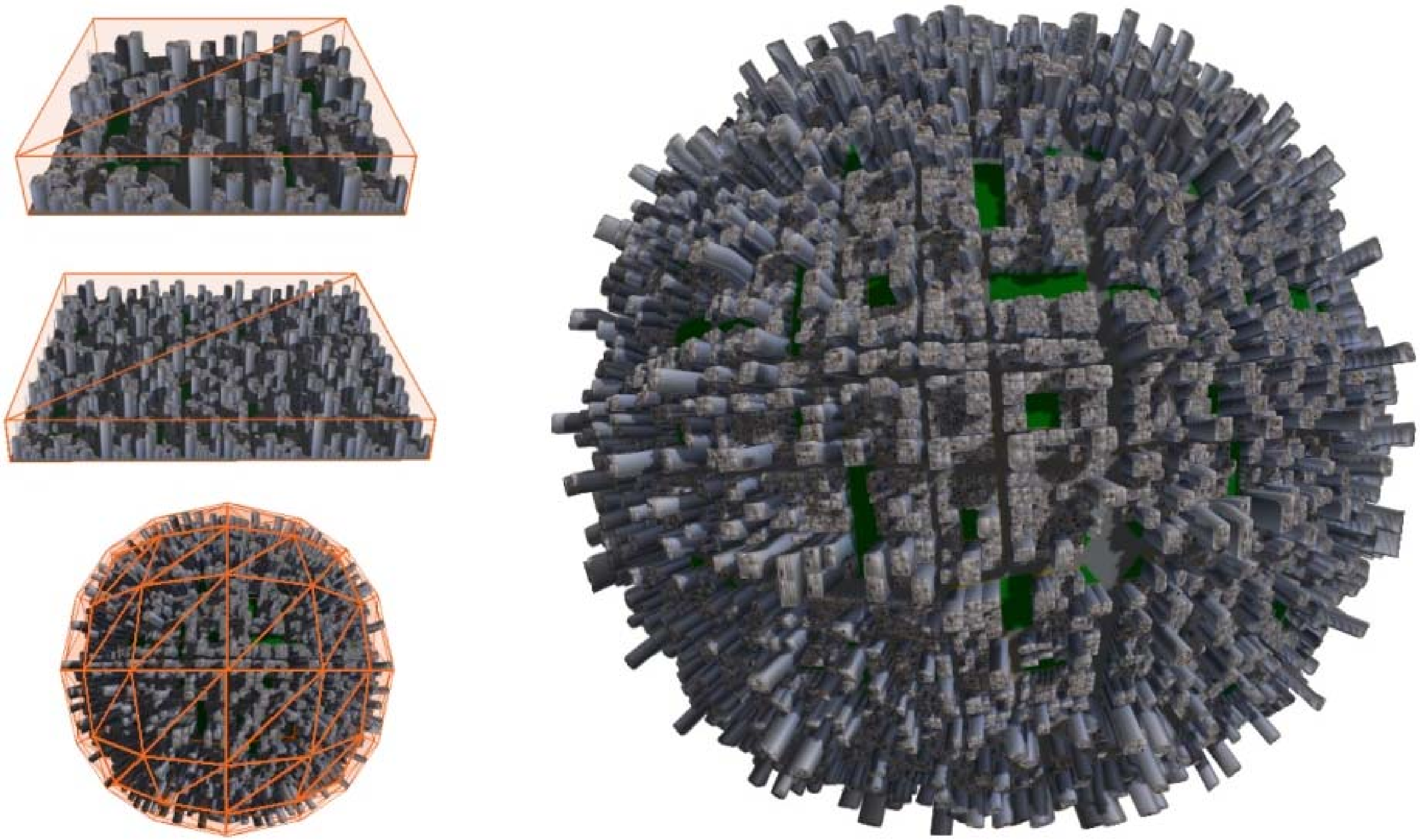
Volumetric Billboards





Decaudin and Neyret "Volumetric Billboards" http://www.antisphere.com/Research/VolumetricBillboards.php

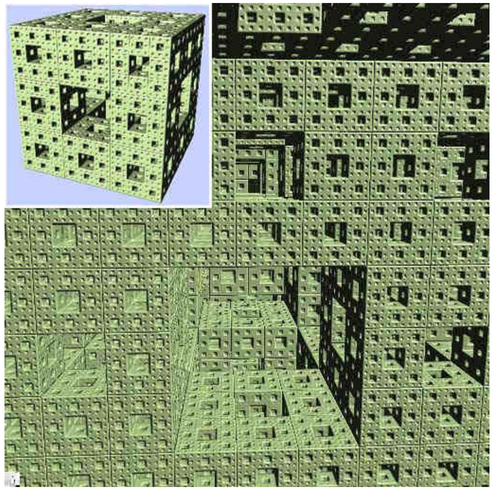
Volumetric Billboards

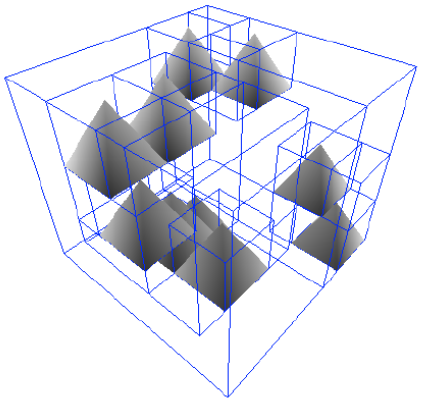


Decaudin and Neyret “Volumetric Billboards” http://www.antisphere.com/Research/VolumetricBillboards.php

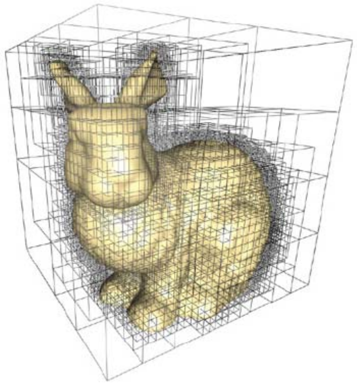
Sparse Voxel Octree

- At high resolutions, voxels can no longer fit on current GPU memory and need to be selectively streamed
- Spatial subdivision hierarchies can be used to speed up redundant ray marching e.g. Empty Space Skipping quickly culls regions of empty (or un-targetted) values
- Can also be used for selective uploading to GPU e.g. Only visible regions





Kd-tree



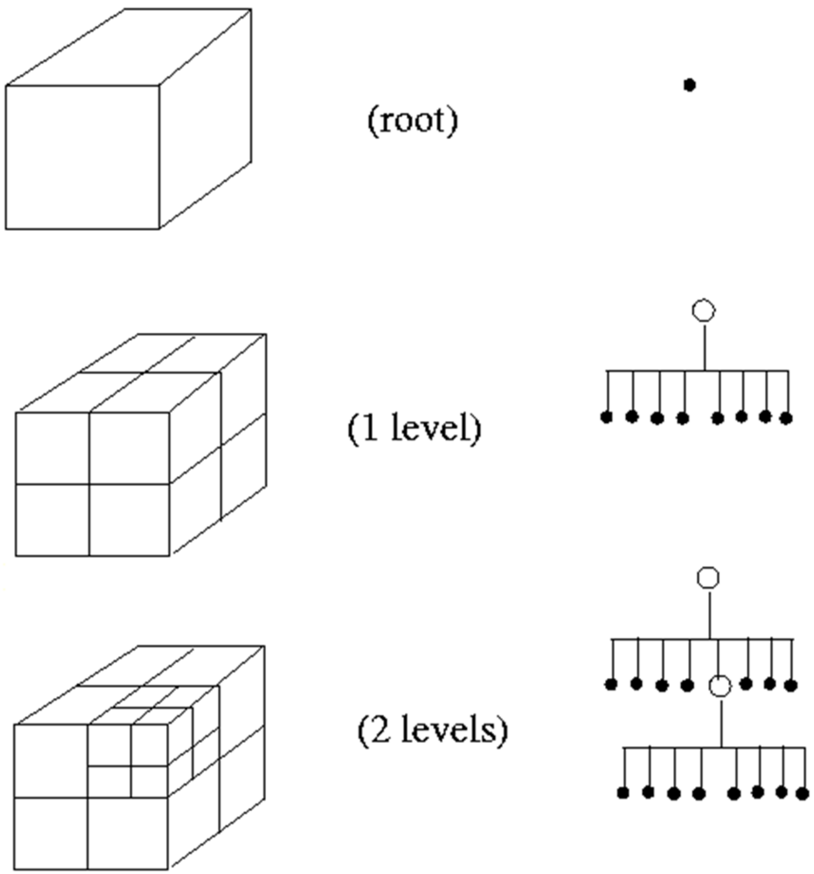
Oct-tree

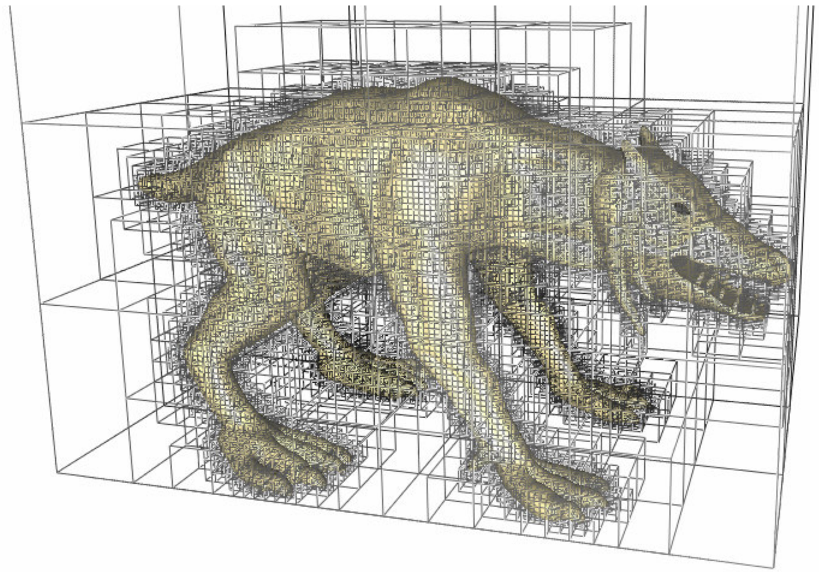
- State-of-the Art: Gigavoxels achieves real-time out-of-core rendering of several billion voxels.
  - N^3 data structure
  - Adaptive data representation
  - Occlusion information

Gigavoxels: http://artis.imag.fr/Publications/2009/CNLE09/

Sparse Voxel Octree

Octree Hierarchy



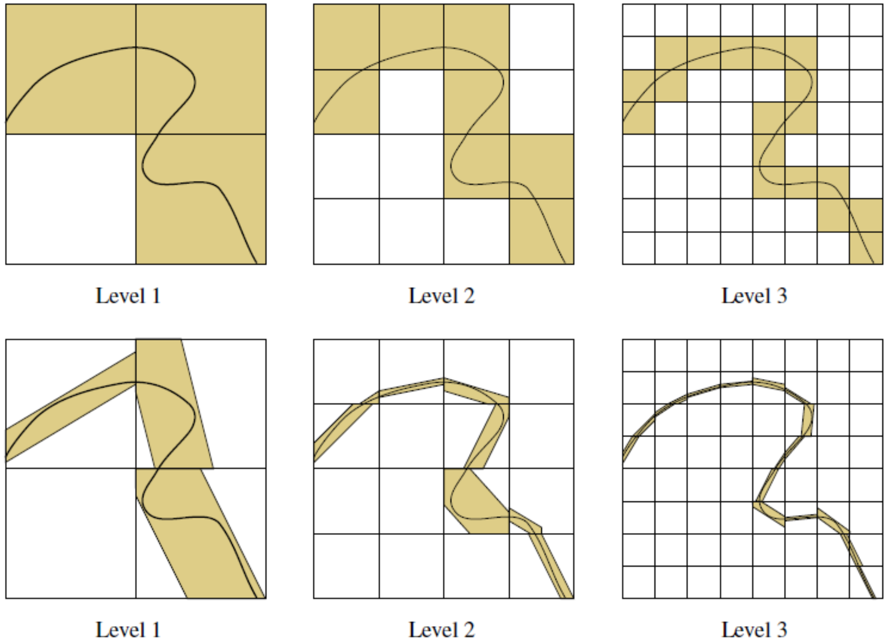


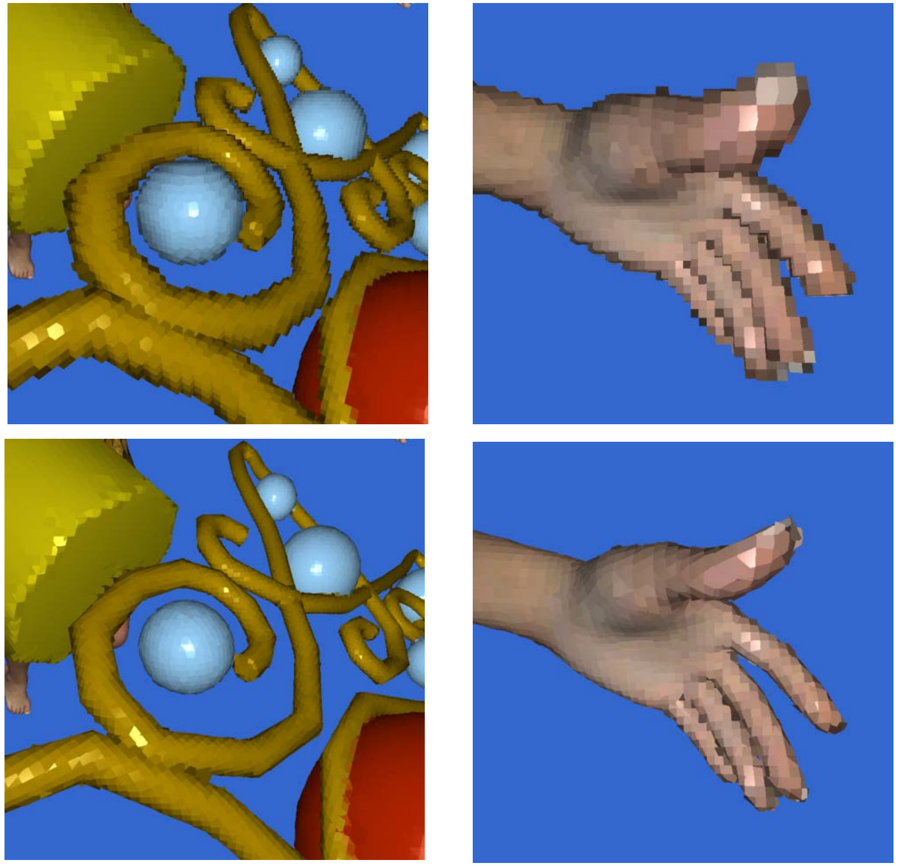
For a **sparse octree** data is stored only around the surface (hull)

Hierarchy can be stored in GPU as **indirection grids**: instead of pointers store indices within textures

For details see: http://lefebvre.sylvain.free.fr/octreetex/octree_textures_on_the_gpu.pdf

SVO Voxel Contours



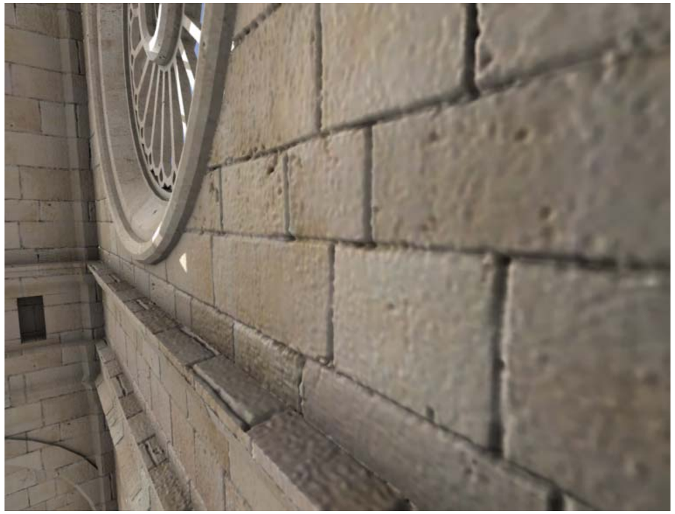


Voxel slab approximates the surface orientation of the object at each voxel better

Can be represented by normal (vec3) and positions in the voxel (2 x ints)

Sparse Voxel Octree

- Store hull-voxels in hierarchical data structure
- Extract data when needed:
  - Traverse tree to required depth based on view: LOD
  - Speed up traversal by quickly skiping non-visible areas: culling
- Advantages over traditional pipeline:
  - Automatic level of detail: geometry and texture at once
  - Colour, displacement maps, normal, BRDF? All in one unified data structure
  - No texture-coordinates required
- With current hardware most implementations are on static scene objects but SVO could be the next big thing



http://www.tml.tkk.fi/~samuli/publications/laine2010tr1_paper.pdf

References

- Simon Green "Volume Rendering for Games" nVidia GDC 2005 presentation
- http://developer.nvidia.com/object/gdc_2005_presentations.html
- Ikits et al "Volume Rendering Techniques" GPU Gems 2. Chapter 39
- http://http.developer.nvidia.com/GPUGems/gpugems_ch39.html

Interactive Entertainment Technology

Further Reading

- **The Advantages Of Sparse Voxel Octrees** – F.Abi-Chahla
  - http://www.tomshardware.com/reviews/voxel-ray-casting,2423-5.html
- **Efficient Sparse Voxel Octrees – Analysis, Extensions, and Implementation** - Samuli Laine Tero Karras (nVidia)
  - http://www.tml.tkk.fi/~samuli/publications/laine2010tr1_paper.pdf
- **Octree Textures on the GPU** - Sylvain Lefebvre, Samuel Hornus, Fabrice Neyret (INRIA)
  - http://lefebvre.sylvain.free.fr/octreetex/octree_textures_on_the_gpu.pdf