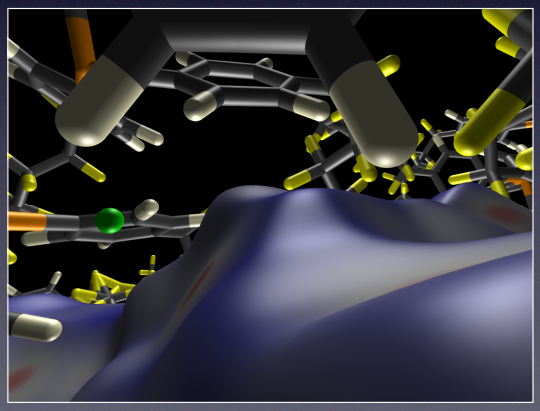
Realtime Raytracing: Applications for Science Visualisation

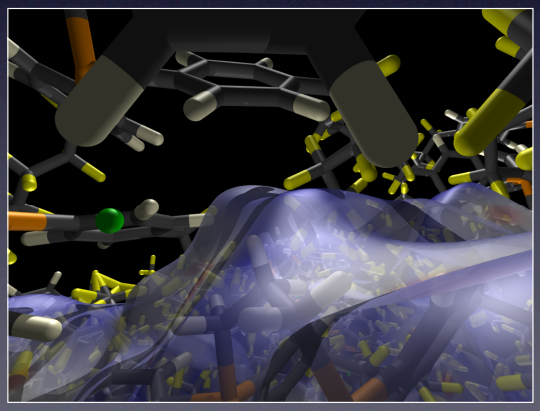
Why do we care and
does it stand a chance?

Paul Bourke
WASP, UWA

Initial comments

- This is not to downplay or criticise the current rasteristaion / scanline / z-buffer algorithms nor the hardware assisted implementations (graphics cards).
- It is an interesting question: "Where would we be today if the energy that has gone into accelerating rasterisation techniques had been equally invested in accelerating raytracing algorithms?"
- Raytracing isn't actually "slow" but it has a high initial overhead. Rasterisation draws all geometry whether it overwrites other geometry (a waste) or not and advanced techniques require multiple passes. Raytracing doesn't require multiple passes and doesn't waste rays.

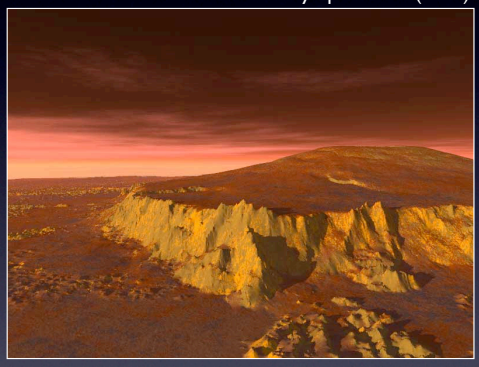




Rendering for Visualisation: characteristics

- Usually have large volumes of multidimensional data.
- Often need to represent high level geometric primitives.
- Approximations are often/usually undesirable.
- It is often improved by enhanced depth cues.
- Can benefit from novel display technologies.

Olympus Mons (Mars)

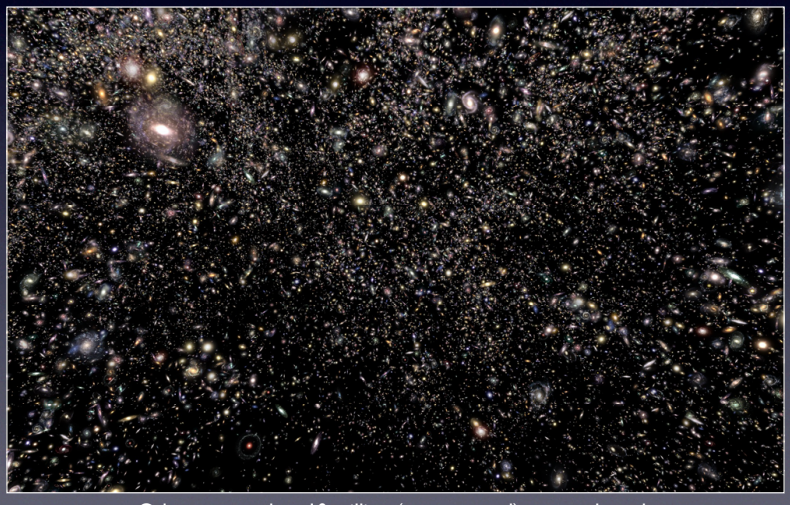


360*180*128*2 = 16 million polygons

- Caveat: Will often be talking in generalisations.
  There are always clever techniques being developed that blur the lines between raytracing and what I will refer in general as rasterisation (or z-buffer) techniques (scanline, OpenGL, Direct3D, PRman, etc).

Raytracing vs z-buffer

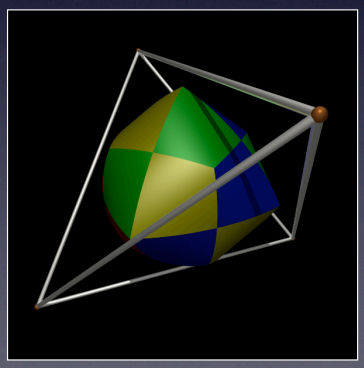
- Data volumes
  - Raytracing as a general rule is less sensitive to the volume of data (pixel bound). Logarithmic dependence on scene complexity vs linear for z-buffer.
  - Intersection tests of bounding volumes is very efficient.
  - Geometry transfer to the card can be the limiting factor, particularly for unstructured data where common tricks can't be efficiently employed to pre-cull the geometry.



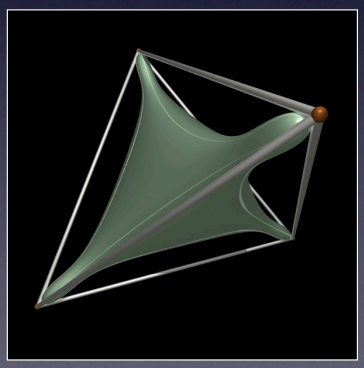
Galaxy survey data, 10 million (unstructured) textured quads

Raytracing vs z-buffer

- High level primitives
  - Raytracing doesn't require decomposition of primitives to planar facets.
  - Facet decomposition introduces a surface approximation.
  - A high level primitive only requires that one can perform a ray intersection and normal calculation.
  - For example a sphere is the fastest primitive for raytracing, N facets for z-buffer.
  - Parametric surfaces, isosurfaces, blobby-molecules ... all rendered directly.
  - Solid constructive geometry can also be rendered directly without any decomposition.



CSG intersection of 4 cylinders



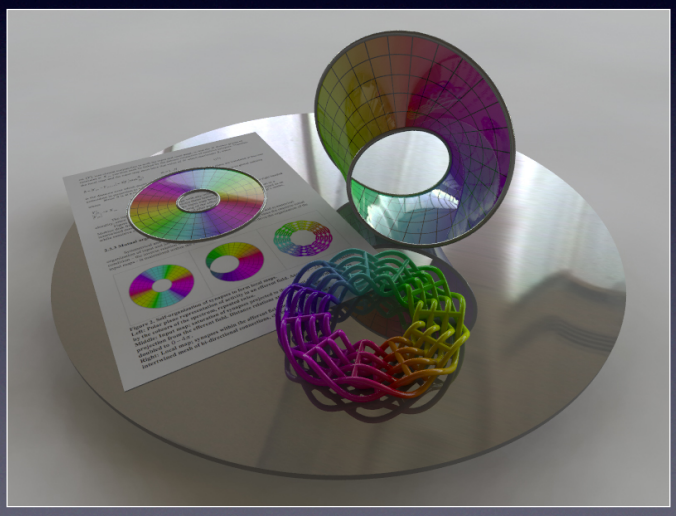
Isosurface

Raytracing vs z-buffer

- Approximations
  - Facet decomposition introduces a surface approximation, important for example when comparing close surfaces for separation distance.
  - Rasterisation generally requires multipass rendering with performance related to the number and complexity of the light sources.
  - Shadow approximations for most rasterisation applications employ approximations, baked on textures, textures based upon lower resolution geometry, etc.

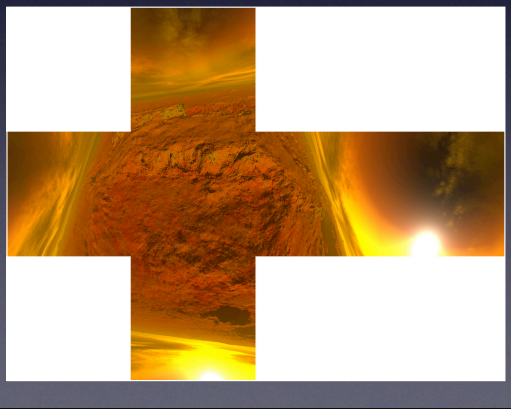
Raytracing vs z-buffer

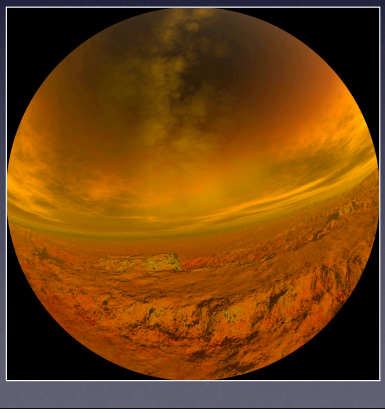
- Improved depth cues from a more capable lighting model
  - Self shadowing, expensive to do properly (especially self shadows) with z-buffer.
  - Correct reflections and refraction standard with raytracing, mostly only created for visual effect rather than correctness with z-buffer.
  - Light scattering and media effects.
  - Radiosity and global illumination (ambient light) calculations greatly improve realism.



Raytracing vs z-buffer

- Novel displays
  - Immersive displays often require more than parallel and perspective projections offered by current raster rendering: fisheye, cylindrical, spherical, omni-directional stereoscopic panoramic projections
  - For example: fisheye or spherical projections require either
    1. multiple passes each with a N x 90 degree perspective projection (N=4 for fisheye)
    2. vertex shader requiring line/facet tessellation (order of magnitude more geometry)
  - Any projection can be created directly in a raytracer, simply involves choosing rays through each pixel in the destination image projection.





How can raytracing possibly compete?

- Shadowing, transparency, reflections, refraction ... can be expensive and often require approximations for z-buffer style rendering, in particular for a generalised application and geometry. A natural consequence of raytracing.
- Improved quality by not needing to reduce higher level geometries to facet approximations.
- Higher level surface primitives for a fraction of the computational cost of facet based versions. Only require an intersection of a ray and normal test for any point on the surface. Particularly interesting for visualisation of mathematics. Spheres are the fastest primitive for a raytracer since it has the simplest intersection test.
- Generally more tunable parameters that affect the speed: antialiasing level, number of secondary rays, number of ray intersections, ray bailout, number of secondary rays, and so on.
- Compared to z-buffer style rendering the performance is comparatively insensitive of the scene complexity, geometry culling to viewport is more expensive than ray and bounding box intersection tests.
- Higher compatibility since no hardware variation and limitations (legacy hardware capabilities) to deal with.

Implementation examples

- Even existing sequential raytracers can readily achieve linear performance improvements by image space partitioning.
- The biggest advances will be made with new raytracing algorithms that explicitly target multiple CPUs (cores), rather than modification of existing largely serial algorithms.
- Open source example: Tachyon (John Stone: Masters student project).
  Threaded and MPI down to each ray cast from the camera.
- OpenRT: Makes significant improvements by parallelising ray bundles to take advantage of coherency of slightly different rays.
- PovRay is currently being rewritten to support multiple core hardware.
- RealStorm: a hardware benchmark employing a raytracing engine.
- Most true realtime attempts achieve their results with significant approximations, to the extent that the results don't necessarily achieve improved visual fidelity over the current state of the art rasterisation techniques.

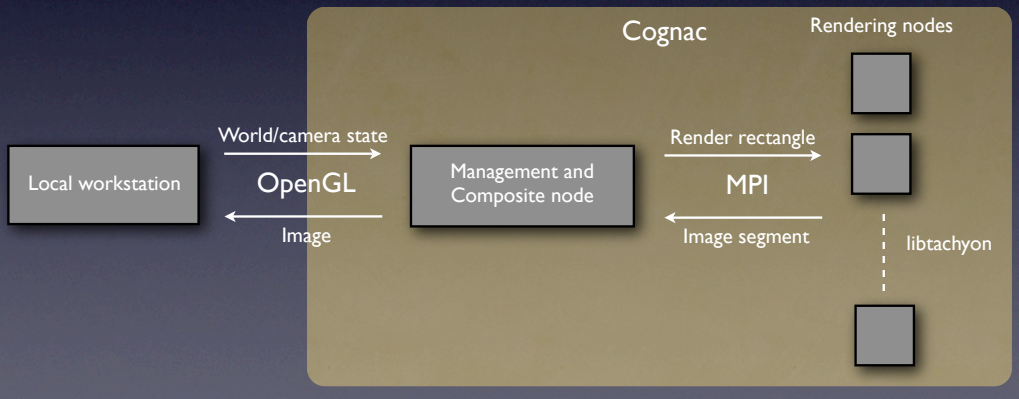
Example for single images

- An example of rendering very expensive models (due to lighting fidelity) on a cluster.
- Render times of tens of hours reduced by partitioning the image into sections.
- The trick is load balancing so one is not waiting at the end for the slowest region.
- Approaches
  - Render single rows at a time. Number of rows >> number of CPUs.
  - Subsample image into a grid. Number of grid cells >> number of CPUs.
  - Estimate complexity across the image (trace lower resolution image) and partition accordingly for full resolution image.

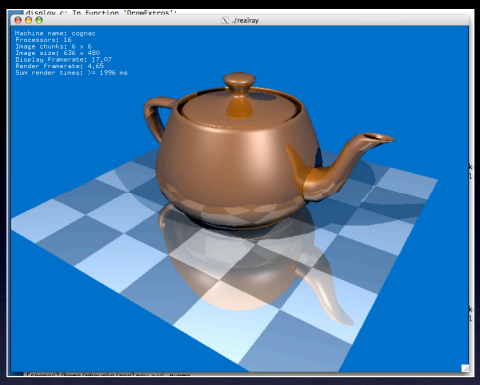
Rob Richens

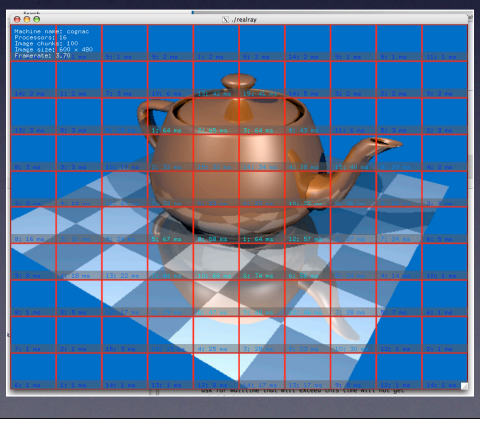
Interactive space partitioning example

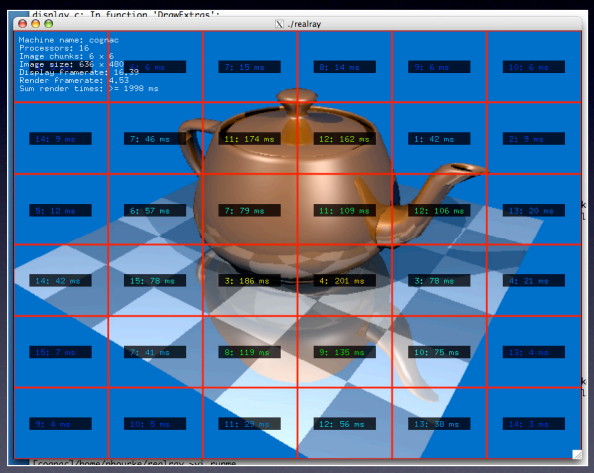
- Significant issue is bandwidth to the workstation. Consider a modest image size 800x600, uncompressed 3 bytes per pixel, at 30fps is 40+ MBytes/sec.
- Used the gridding of image approach, simple, allows the grid to be readily adjusted for tuning, still expect the number of grid cells to be much larger than the number of CPUs. In the cycle of a frame one CPU may render multiple cells while another may only render one or a few.
- Simple architecture for rendering on COGNAC



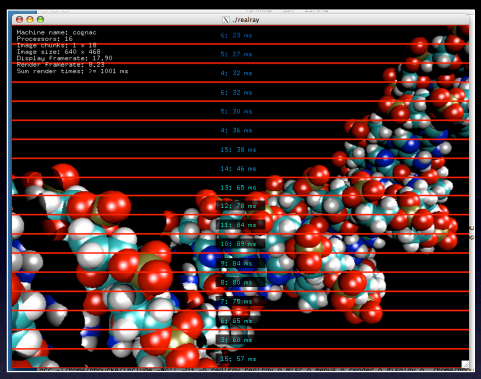
Interactive space partitioning example 1

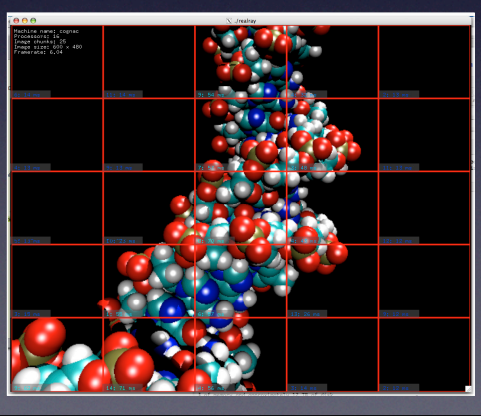


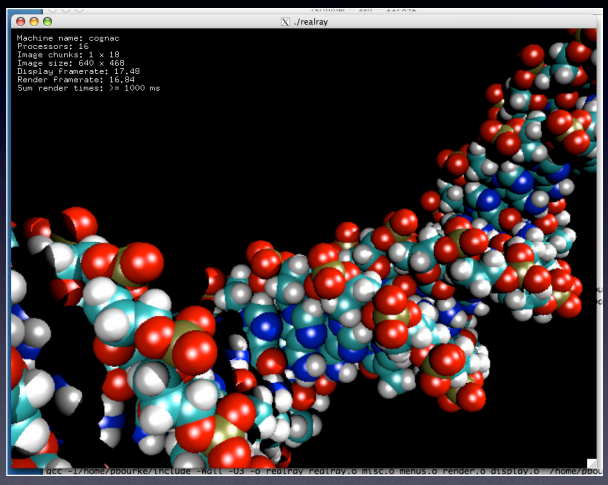




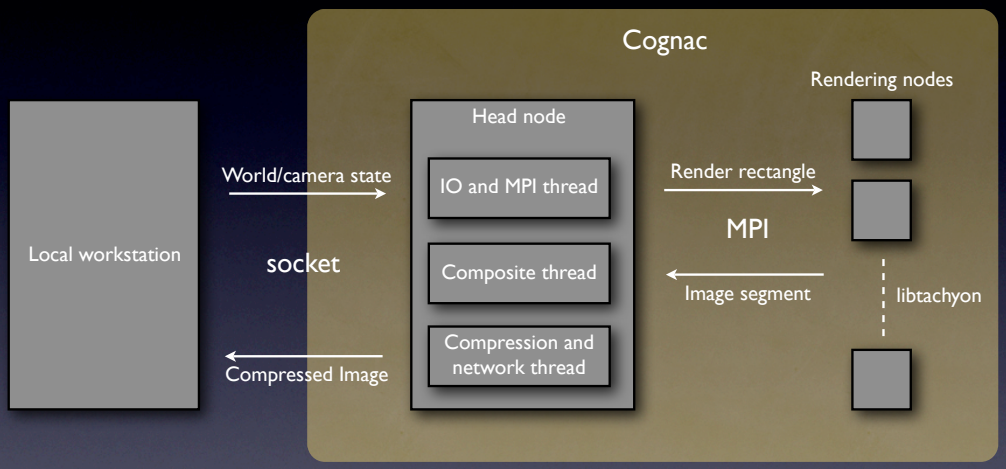
Interactive space partitioning example 2







Next implementation



Questions?

Recommended reading.
Interactive Ray Tracing: the replacement of rasterization.
A.J. van der Ploeg