Order-Independent Transparency

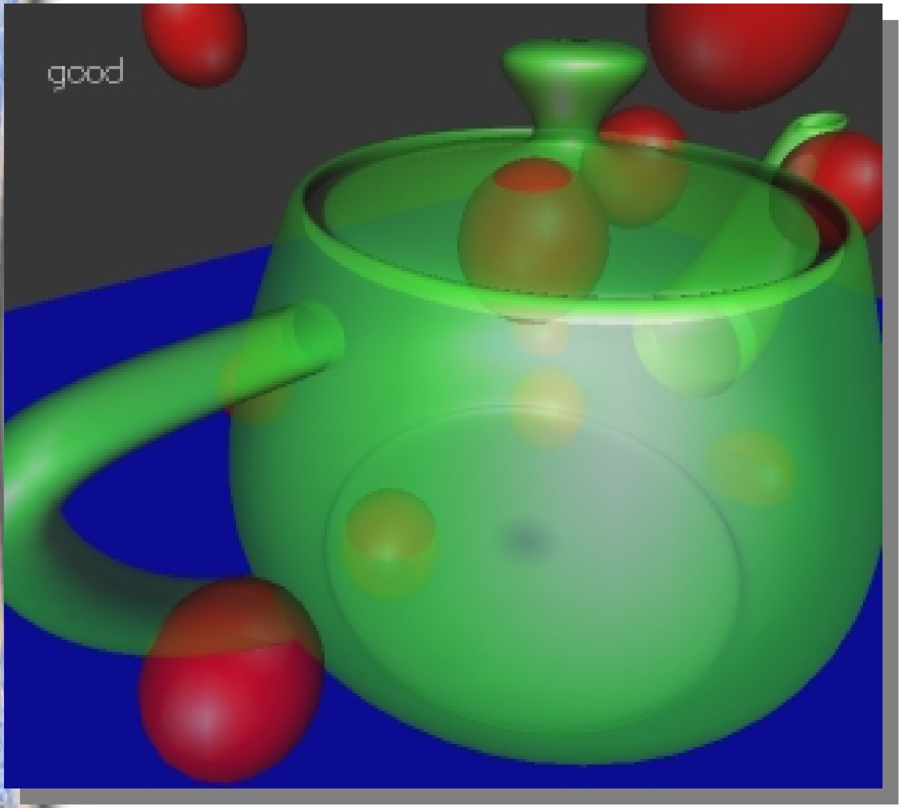
Cass Everitt
NVIDIA Corporation
cass@nvidia.com

Overview

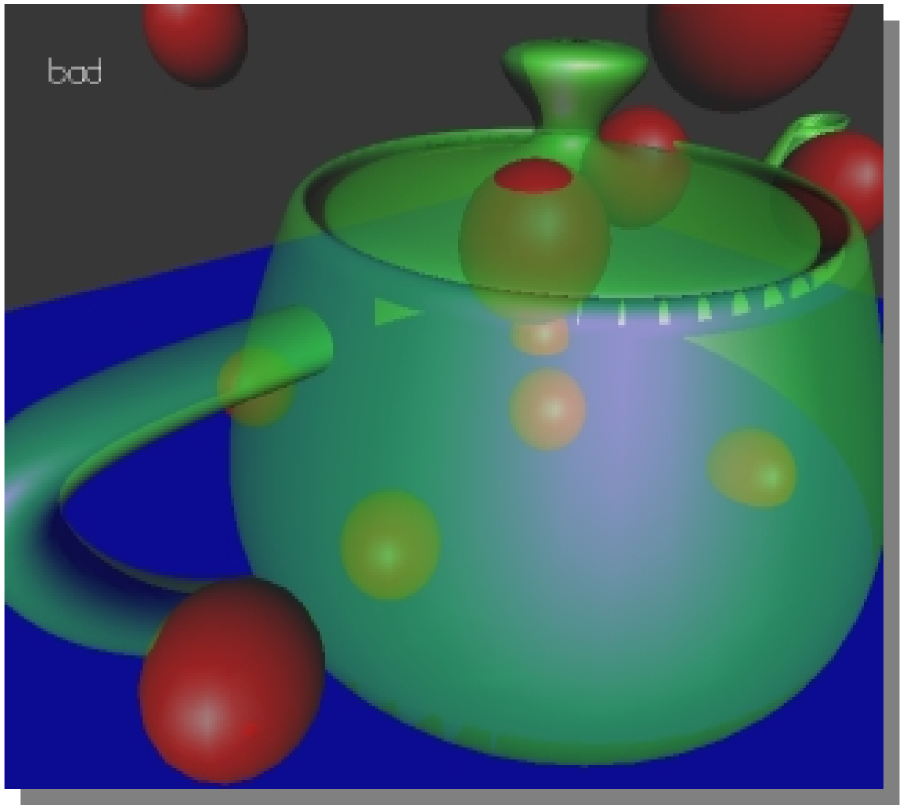
- Why is correct transparency hard?
- Depth peeling
  - Two depth buffers
- Enter the shadow map
  - Precision/invariance issues
  - Depth replace texture shader
- Blending the layers
- Other applications

Can’t just glEnable(GL_BLEND)…

Good Transparency **Bad Transparency**



<u>with</u> OIT



<u>without</u> OIT

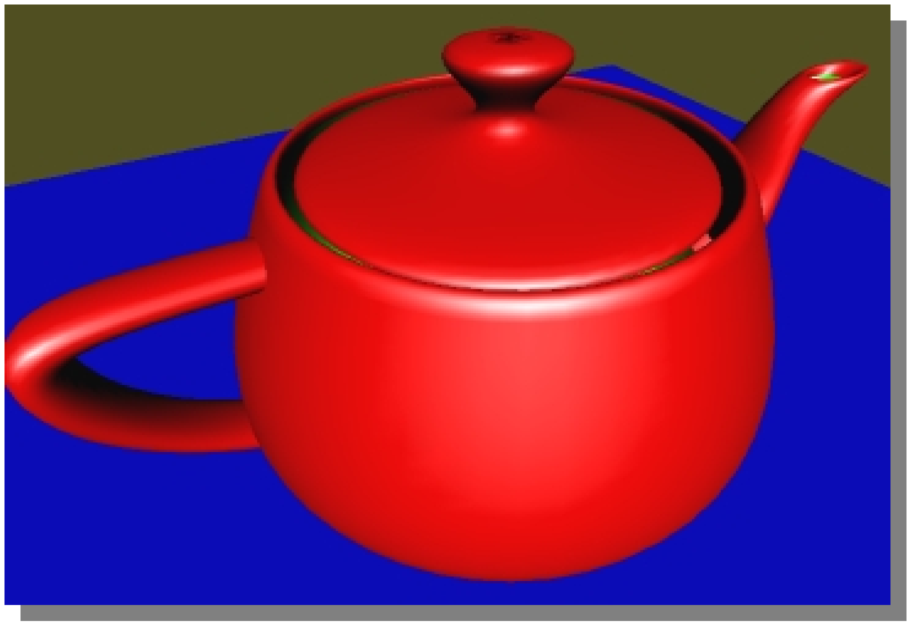
Why is correct transparency hard?

- Most hardware does object-order rendering
- Correct transparency requires sorted traversal
  - Have to render polygons in sorted order
    - Not very convenient
  - Polygons can’t intersect
  - Lot of extra application work
    - Especially difficult for dynamic scene databases

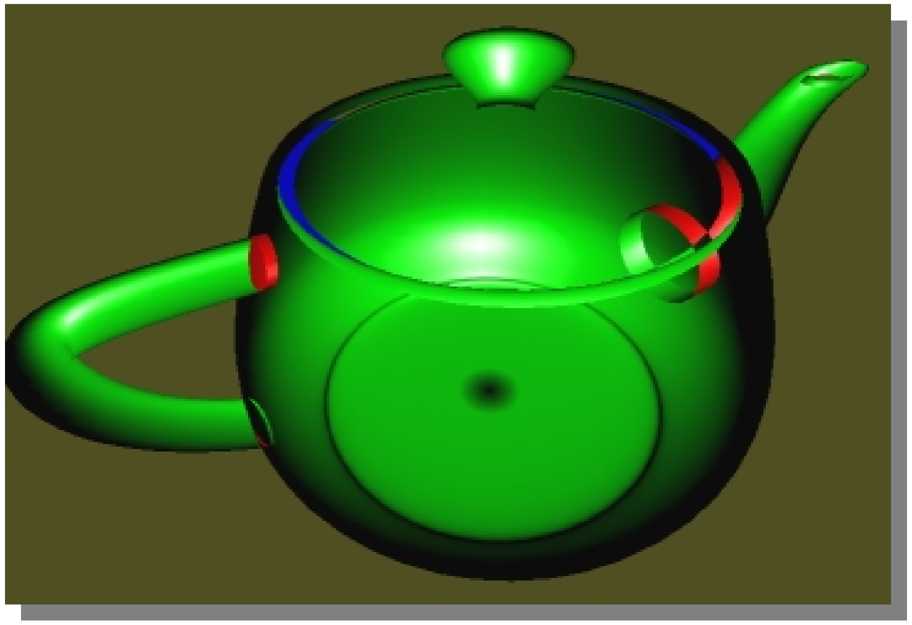
Depth Peeling

- The algorithm uses an “implicit sort” to extract multiple depth layers
  - First pass render finds front-most fragment color/depth
  - Each successive pass render finds (extracts) the fragment color/depth for the next-nearest fragment on a per pixel basis
  - Use dual depth buffers to compare previous nearest fragment with current
  - Second “depth buffer” used for comparison (read only) from texture [more on this later]

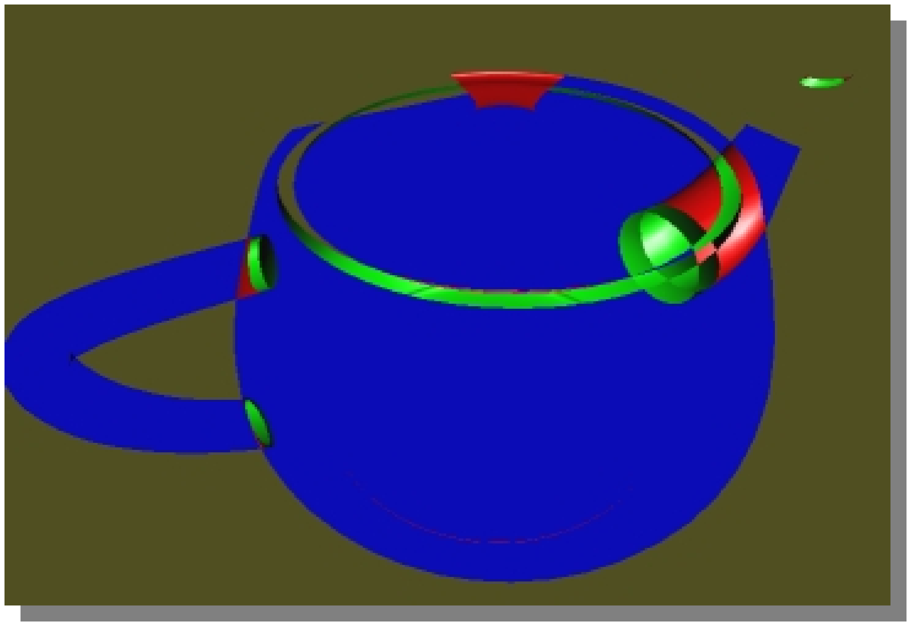
Layer 0



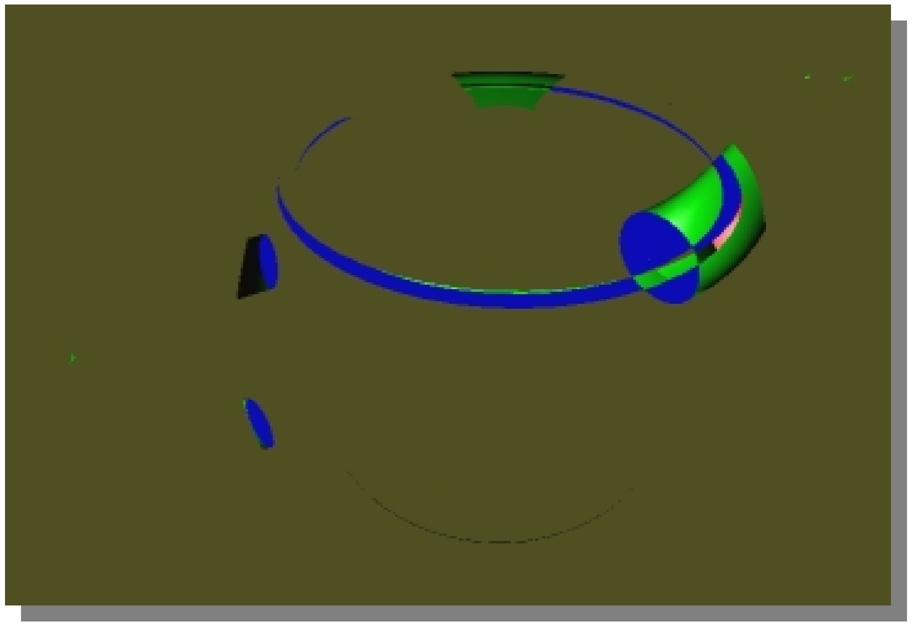
Layer 1



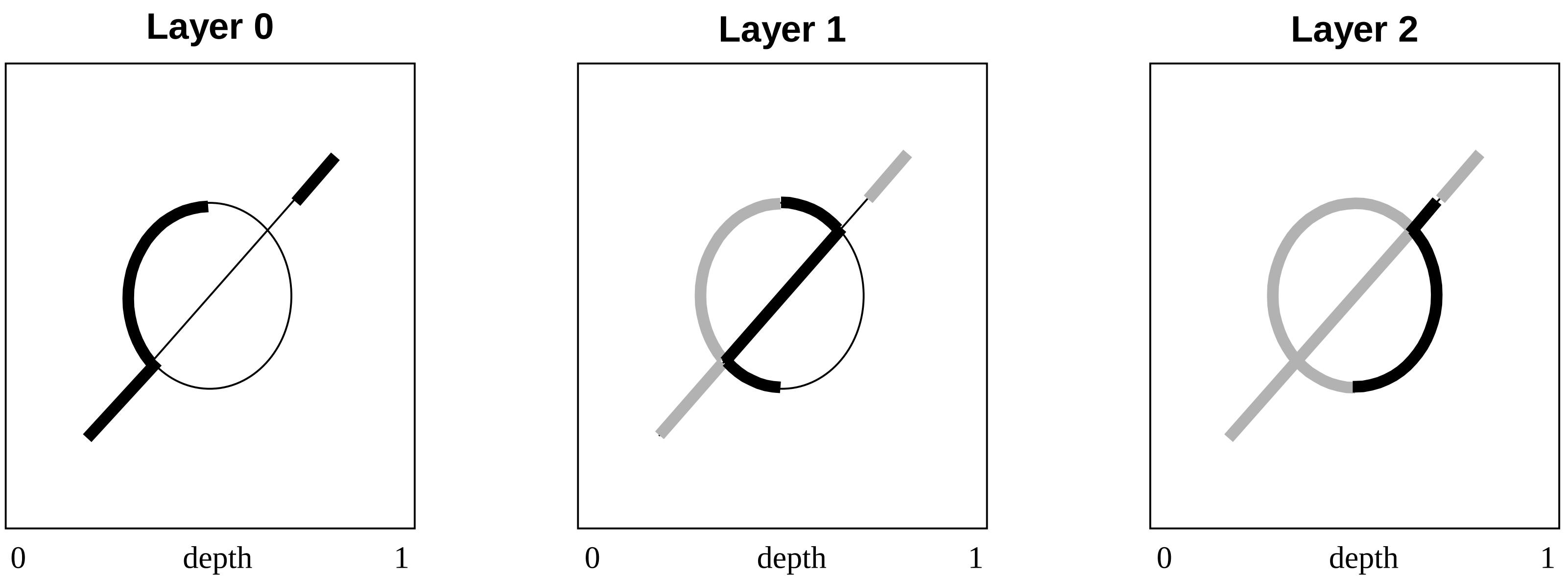
Layer 2



Layer 3



Cross-section view of depth peeling



Depth peeling strips away depth layers with each successive pass. The frames above show the frontmost (leftmost) surfaces as bold black lines, hidden surfaces as thin black lines, and "peeled away" surfaces as light grey lines.

Dual Depth Buffer Pseudo-code

```
for ( i = 0; i < num_passes; i++ )
{
   clear color buffer
   depth unit 0:
       if(i == 0) { disable depth test }
       else       { enable depth test  }
       bind depth buffer (i % 2)
       disable depth writes  /* read-only depth test */
       set depth func to GREATER
   depth unit 1:
       bind depth buffer ((i+1) % 2)
       clear depth buffer
       enable depth writes;
       enable depth test;
       set depth func to LESS
   render scene
   save color buffer RGBA as layer i
}
```

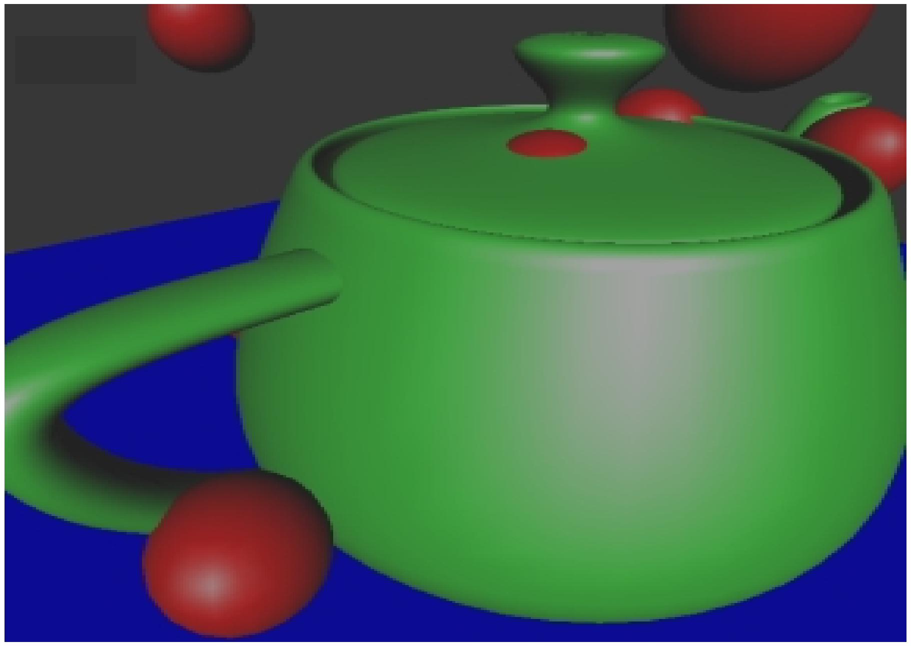
Implementation

- There is no “dual depth buffer” extension to OpenGL, so what can we do?
- Just need one depth test with writeable depth buffer – the other can be read-only
  - Shadow mapping *is* a read-only depth test!
    - Depth test can have an arbitrary camera location
    - Other interesting uses for clip volumes
  - Fast copies make this proposition reasonable
  - Copies will be unnecessary in the future…

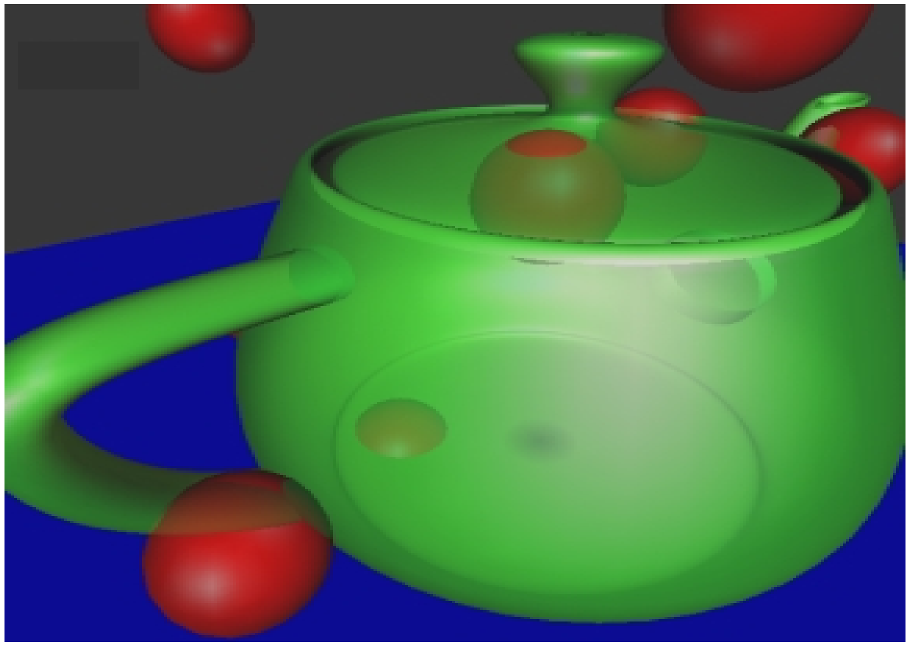
Precision / Invariance issues

- Using shadow mapping hardware introduces precision and invariance issues
  - depth rasterization usually just needs to match output depth buffer precision, and requires no perspective correction
  - Texture hardware requires perspective correction and projection at high precision
  - Making things match would be difficult without the DEPTH_REPLACE texture shader
    - Computes with texture hardware at texture precision
    - Solves invariance problems at some extra expense
    - Will be cheaper in the future…

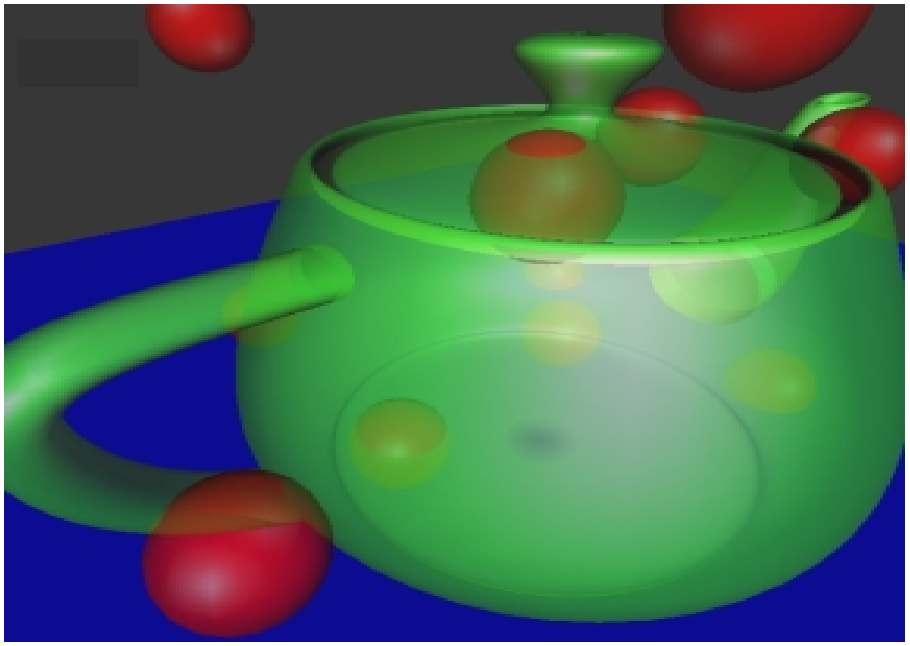
1 layer



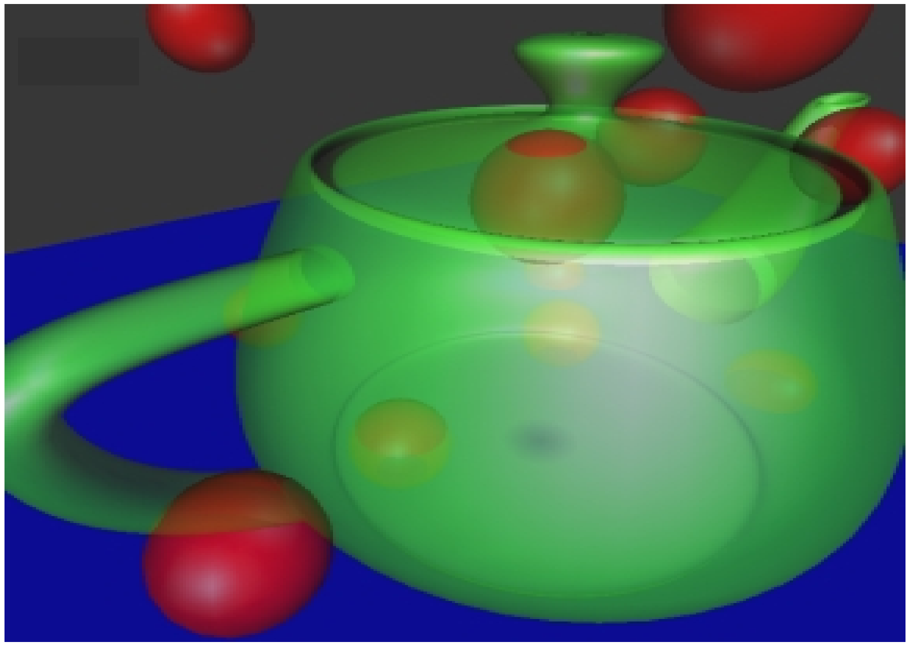
2 layers



3 layers



4 layers



Compositing

- Each time we peel, we capture the RGBA, then as a final step, we blend all the layers together from back to front
  - Opaque fragments completely overwrite previous transparent ones
  - Could peel from back to front and start with all fully opaque objects
    - Fewer layers required
    - Would be useful with occlusion / visibility test
    - Must peel all layers to avoid blending artifacts
    - Variations on this approach would still allow early exit (buffer_region extension)

Conclusions

- Results are nice!
  - Get correct transparency without invasive changes to internal data structures
  - Can be “bolted on” to existing CAD/CAM apps
  - Requires n scene traversals for n correctly sorted depths
    - $n = 4$ is often quite satisfactory (see previous slide)
- Depth Peeling instrumental in “Woo shadowmaps”
  - Other uses?
- Shadow maps are for more than shadows!

Questions?

- cass@nvidia.com
- http://www.nvidia.com/developer

Thanks for attending