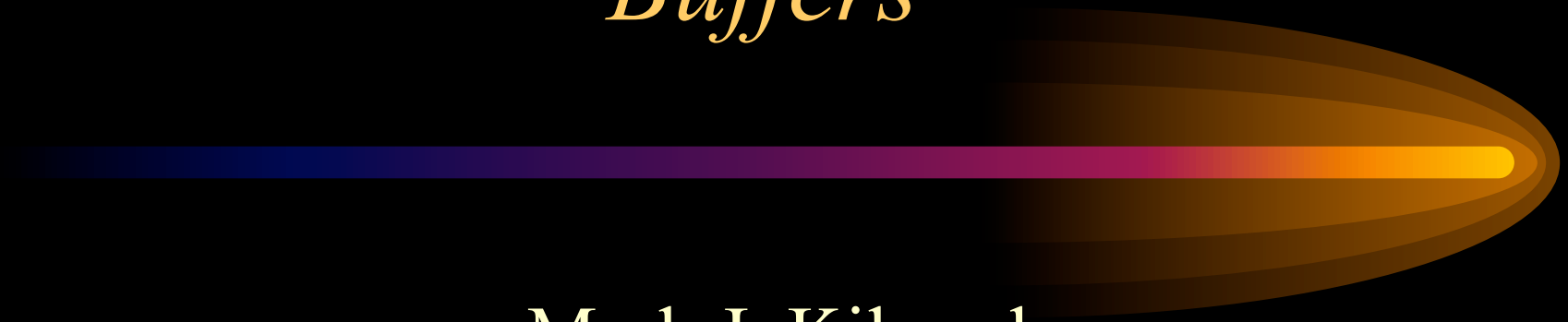
Creating Reflections and Shadows Using Stencil Buffers



Mark J. Kilgard
NVIDIA Corporation

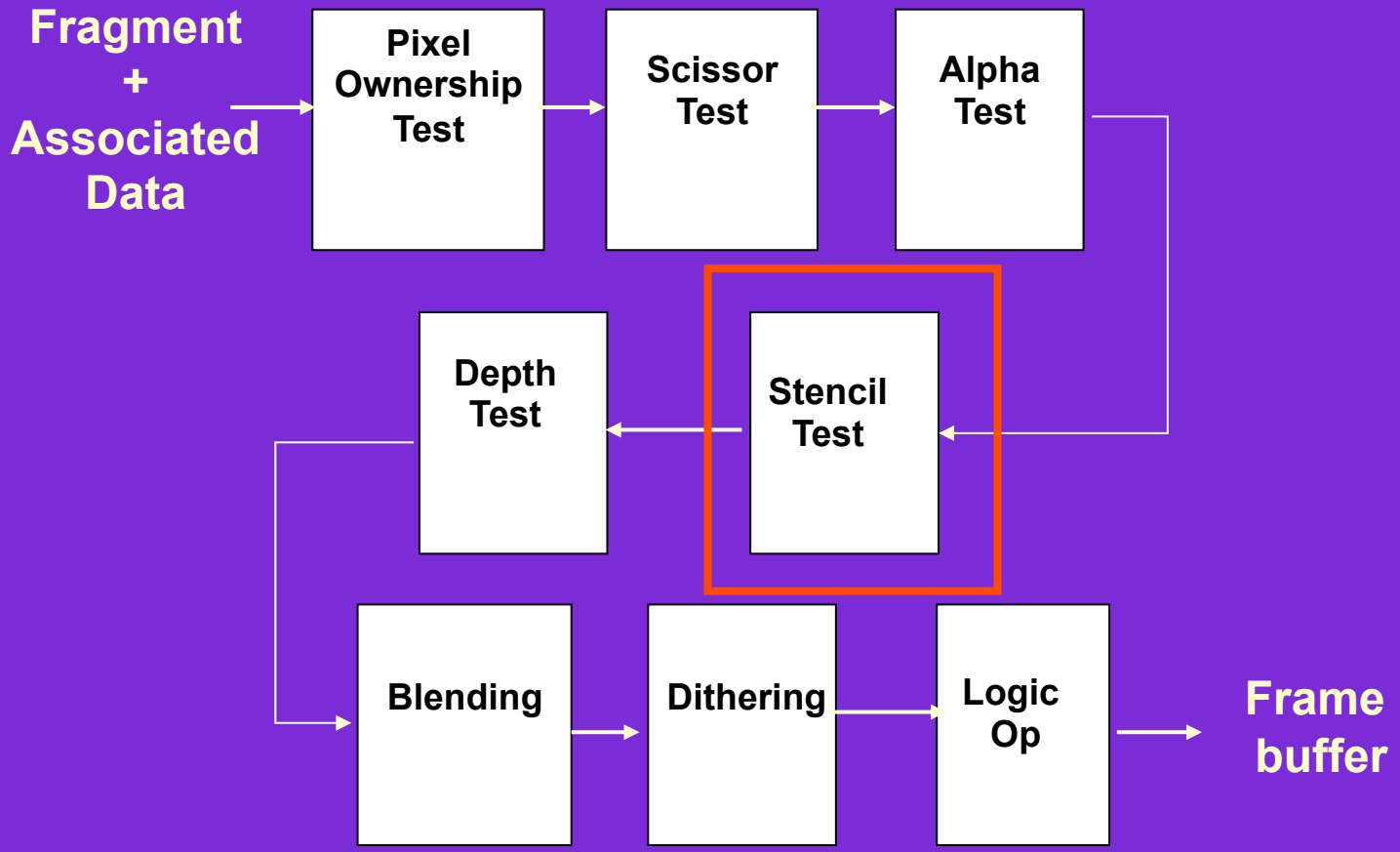
Stencil Testing

- Now broadly supports by both major APIs
  - OpenGL
  - DirectX 6
- RIVA TNT and other consumer cards now supporting full 8-bit stencil
- Opportunity to achieve new cool effects and improve scene quality

What is Stenciling?

- Per-pixel test, similar to depth buffering.
- Tests against value from stencil buffer; rejects fragment if stencil test fails.
- Distinct stencil operations performed when
  - Stencil test fails
  - Depth test fails
  - Depth test passes
- Provides fine grain control of pixel update

Per-Fragment Pipeline



OpenGL API

- glEnable/glDisable(GL_STENCIL_TEST);
- glStencilFunc(function, reference, mask);
- glStencilOp(stencil_fail,
  depth_fail, depth_pass);
- glStencilMask(mask);
- glClear(… | GL_STENCIL_BUFFER_BIT);

Request a Stencil Buffer

- If using stencil, request sufficient bits of stencil
- Implementations may support from zero to 32 bits of stencil
- 8, 4, or 1 bit are common possibilities
- Easy for GLUT programs:

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |
                    GLUT_DEPTH | GLUT_STENCIL);
glutCreateWindow("stencil example");
```

Stencil Test

- Compares reference value to pixel's stencil buffer value
- Same comparison functions as depth test:
  - NEVER, ALWAYS
  - LESS, LEQUAL
  - GREATER, GEQUAL
  - EQUAL, NOTEQUAL
- Bit mask controls comparison
  ((ref & mask) op (svalue & mask))

Stencil Operations

- Stencil side effects of
  - Stencil test fails
  - Depth test fails
  - Depth test passes
- Possible operations
  - Increment, Decrement (saturates)
  - Increment, Decrement (wrap, DX6 option)
  - Keep, Replace
  - Zero, Invert
- Way stencil buffer values are controlled

Stencil Write Mask

- Bit mask for controlling write back of stencil value to the stencil buffer
- Applies to the clear too!

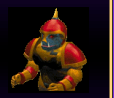
- Stencil compare & write masks allow stencil values to be treated as sub-fields

DX6 Direct3D API

- ***Identical functionality as OpenGL***
- SetRenderState controls update
  - D3DRENDERSTATE_STENCILENABLE
  - D3DRENDERSTATE_STENCILFUNC
  - D3DRENDERSTATE_STENCILREF
  - D3DRENDERSTATE_STENCILMASK
  - D3DRENDERSTATE_STENCILWRITEMASK
  - D3DRENDERSTATE_STENCILFAIL
  - D3DRENDERSTATE_STENCILZFAIL
  - D3DRENDERSTATE_STENCILPASS

DX6 Issues

- Be sure to query capability bits!
- Involves creating and attaching a depth/ stencil surface



Performance

- With today's 32-bit graphics accelerator modes, 24-bit depth and 8-bit stencil packed in *same* memory word
- RIVA TNT is an example
- Performance implication:

  if using depth testing, stenciling is at NO PENALTY

Repeating that!

- On card like RIVA TNT2 in 32-bit mode

  if using depth testing, stenciling has NO PENALTY

- Do not treat stencil as “expensive” -- in fact, treat stencil as “free” when already depth testing

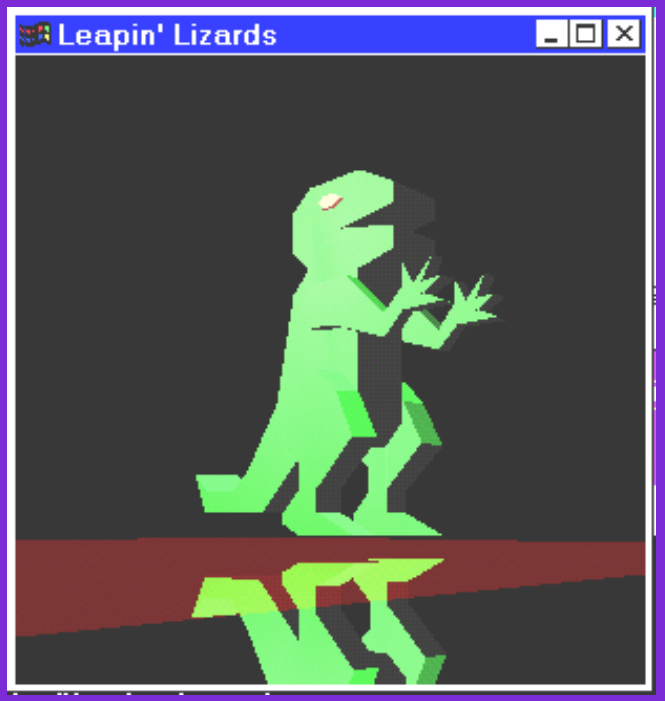
Pseudo Global Lighting Effects

- OpenGL's light model is a "local" model
  - Light source parameters
  - Material parameters
  - Nothing else enters the equation
- Global illumination is fancy word for real-world light interactions
  - Shadows, reflections, refractions, radiosity, etc.
- Pseudo global lighting is about clever hacks

Planar Reflections

Dinosaur is reflected by the planar floor.
Easy hack, draw dino twice, second time has `glScalef(1,-1,1)` to reflect through the floor

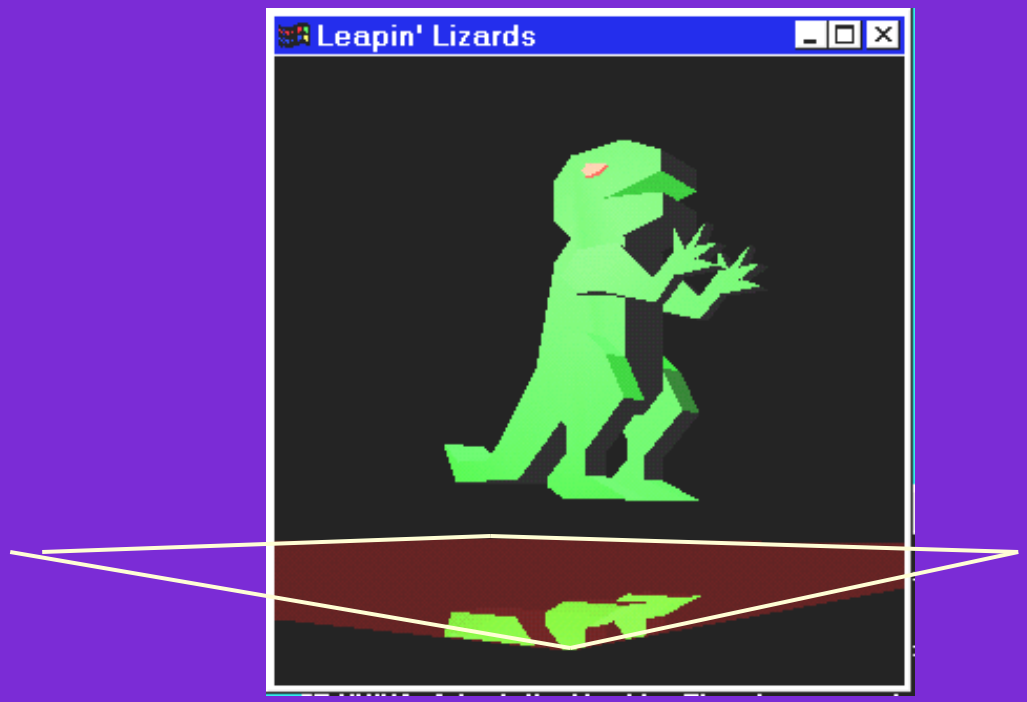
Compare Two Versions



Good. **Bad.**

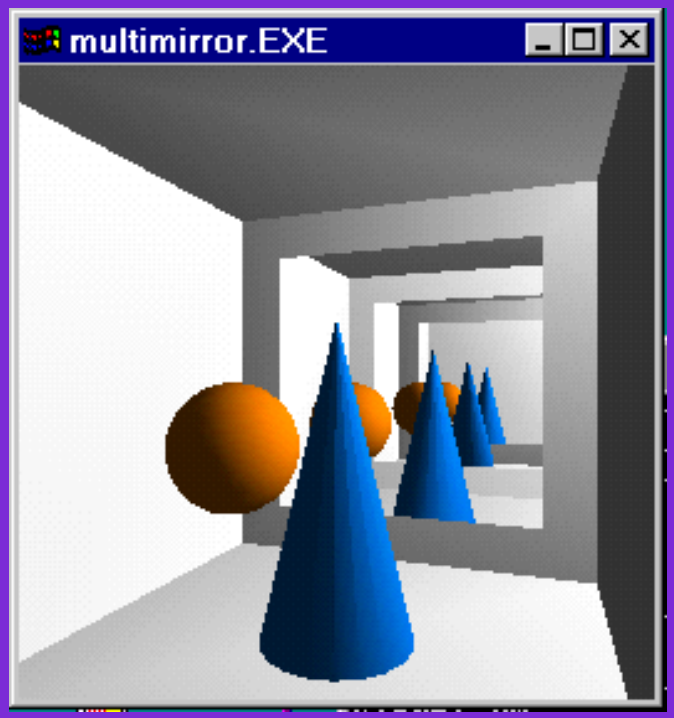
Notice right image's reflection falls off the floor!

Stencil Maintains the Floor



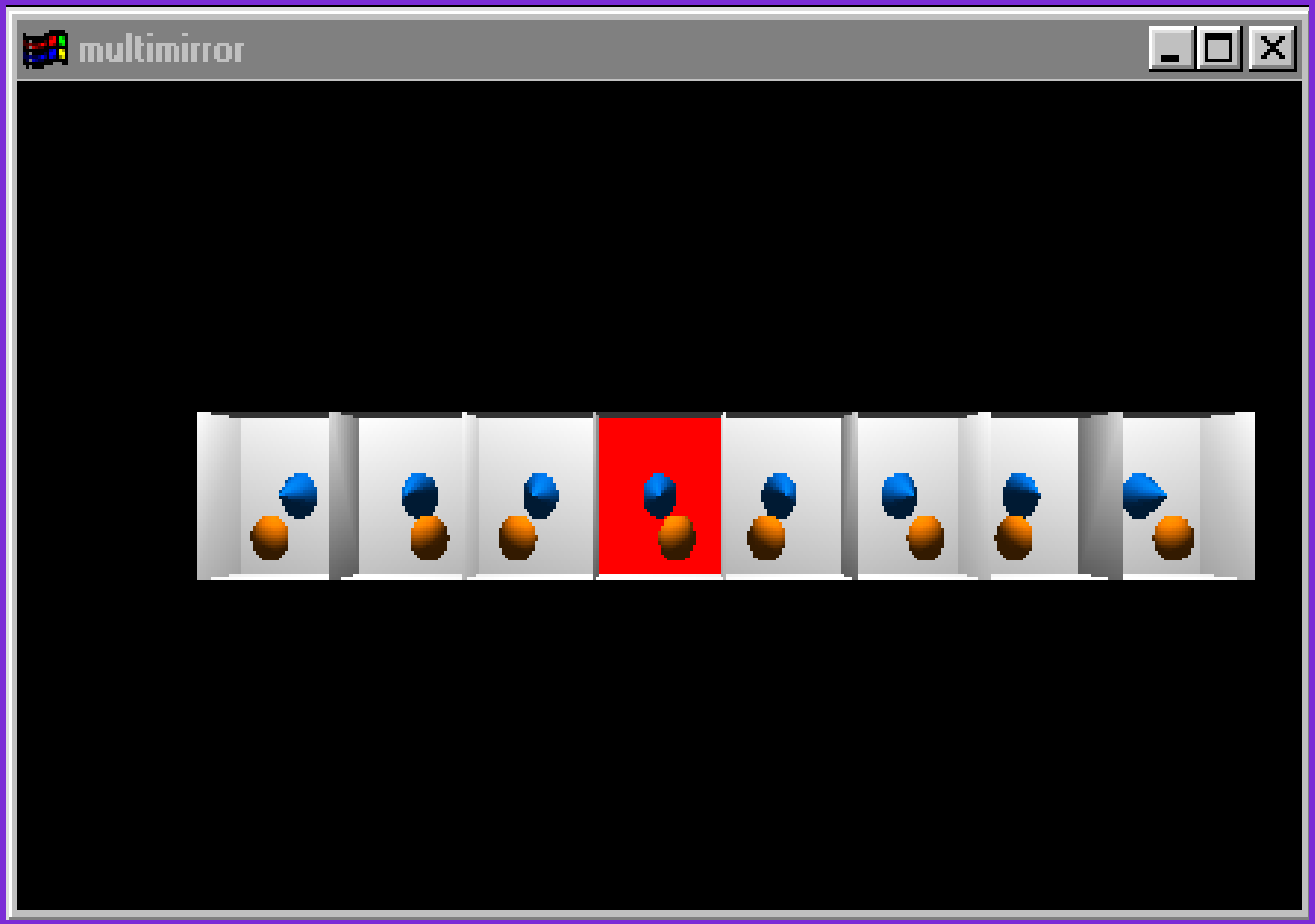
Clear stencil to zero.
Draw floor polygon with stencil set to one.
Only draw reflection where stencil is one.

Recursive Planar Mirrors



Basic idea of planar reflections can be applied recursively. Requires more stencil bits.

The Trick (bird's eye view)



Next: Planar Shadows

Shadow is projected into the plane of the floor.

Constructing a Shadow Matrix

```
void shadowMatrix(GLfloat shadowMat[4][4], GLfloat groundplane[4], GLfloat lightpos[4])
{
  GLfloat dot;
  /* Find dot product between light position vector and ground plane normal. */
  dot = groundplane[X] * lightpos[X] +
   groundplane[Y] * lightpos[Y] +
   groundplane[Z] * lightpos[Z] +
   groundplane[W] * lightpos[W];
  shadowMat[0][0] = dot - lightpos[X] * groundplane[X];
  shadowMat[1][0] = 0.f - lightpos[X] * groundplane[Y];
  shadowMat[2][0] = 0.f - lightpos[X] * groundplane[Z];
  shadowMat[3][0] = 0.f - lightpos[X] * groundplane[W];
  shadowMat[X][1] = 0.f - lightpos[Y] * groundplane[X];
  shadowMat[1][1] = dot - lightpos[Y] * groundplane[Y];
  shadowMat[2][1] = 0.f - lightpos[Y] * groundplane[Z];
  shadowMat[3][1] = 0.f - lightpos[Y] * groundplane[W];
  shadowMat[X][2] = 0.f - lightpos[Z] * groundplane[X];
  shadowMat[1][2] = 0.f - lightpos[Z] * groundplane[Y];
  shadowMat[2][2] = dot - lightpos[Z] * groundplane[Z];
  shadowMat[3][2] = 0.f - lightpos[Z] * groundplane[W];
  shadowMat[X][3] = 0.f - lightpos[W] * groundplane[X];
  shadowMat[1][3] = 0.f - lightpos[W] * groundplane[Y];
  shadowMat[2][3] = 0.f - lightpos[W] * groundplane[Z];
  shadowMat[3][3] = dot - lightpos[W] * groundplane[W];
}
```

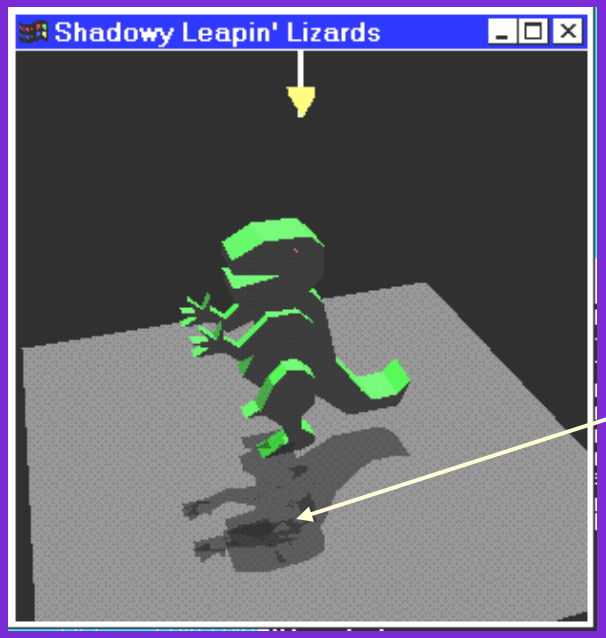
How to Render the Shadow

```
/* Render 50% black shadow color on top of whatever
   the floor appearance is. */
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,
    GL_ONE_MINUS_SRC_ALPHA);
glDisable(GL_LIGHTING);  /* Force the 50% black. */
glColor4f(0.0, 0.0, 0.0, 0.5);

glPushMatrix();
  /* Project the shadow. */
  glMultMatrixf((GLfloat *) floorShadow);
  drawDinosaur();
glPopMatrix();
```

Note Quite So Easy (1)

Without stencil to avoid double blending of the shadow pixels:

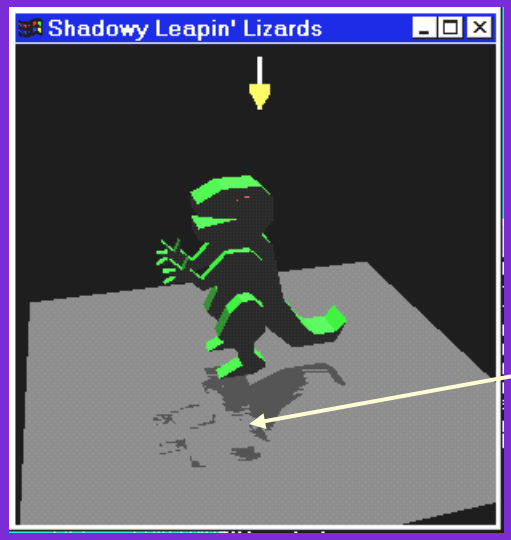


Notice darks spots on the planar shadow.

Solution: Clear stencil to zero. Draw floor with stencil of one. Draw shadow if stencil is one. If shadow's stencil test passes, set stencil to two. No double blending.

Note Quite So Easy (2)

There's still another problem even if using stencil to avoid double blending.



depth buffer Z fighting artifacts

Shadow fights with depth values from the floor plane. Use polygon offset to raise shadow polygons slightly in Z.

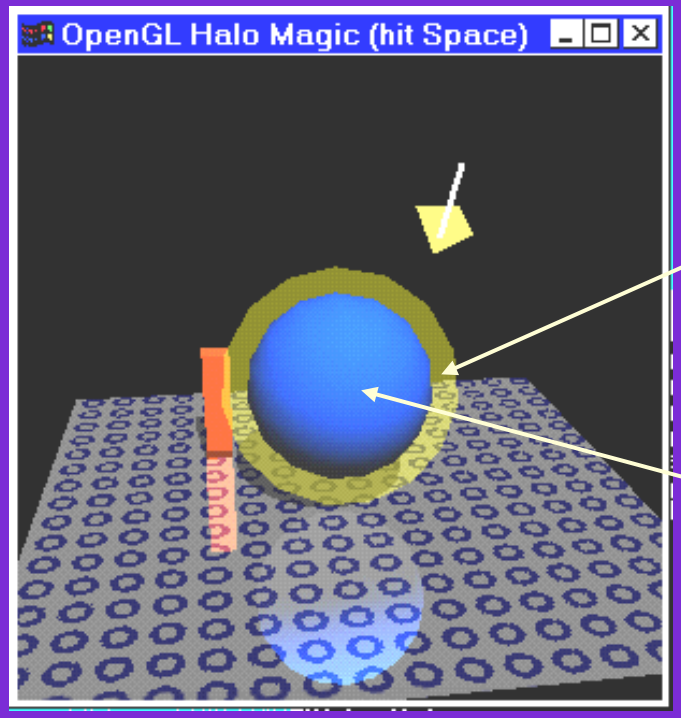
Everything All At Once

Lighting, texturing, planar shadows, and planar reflections all at one time. Stencil & polygon offset eliminate aforementioned artifacts.

Pseudo Global Lighting

- Planar reflections and shadows add more than simplistic local lighting model
- Still not really global
  - Techniques more about hacking common cases based on knowledge of geometry
  - Not really modeling underlying physics of light
- Techniques are “multipass”
  - Geometry is rendered multiple times to improve the rendered visual quality

Bonus Stenciled Halo Effect



Halo is blended with objects behind haloed object.

Halo does not obscure or blend with the haloed object.

Clear stencil to zero. Render object, set stencil to one where object is. Scale up object with glScalef. Render object again, but not where stencil is one.

Stenciled Shadow Volumes

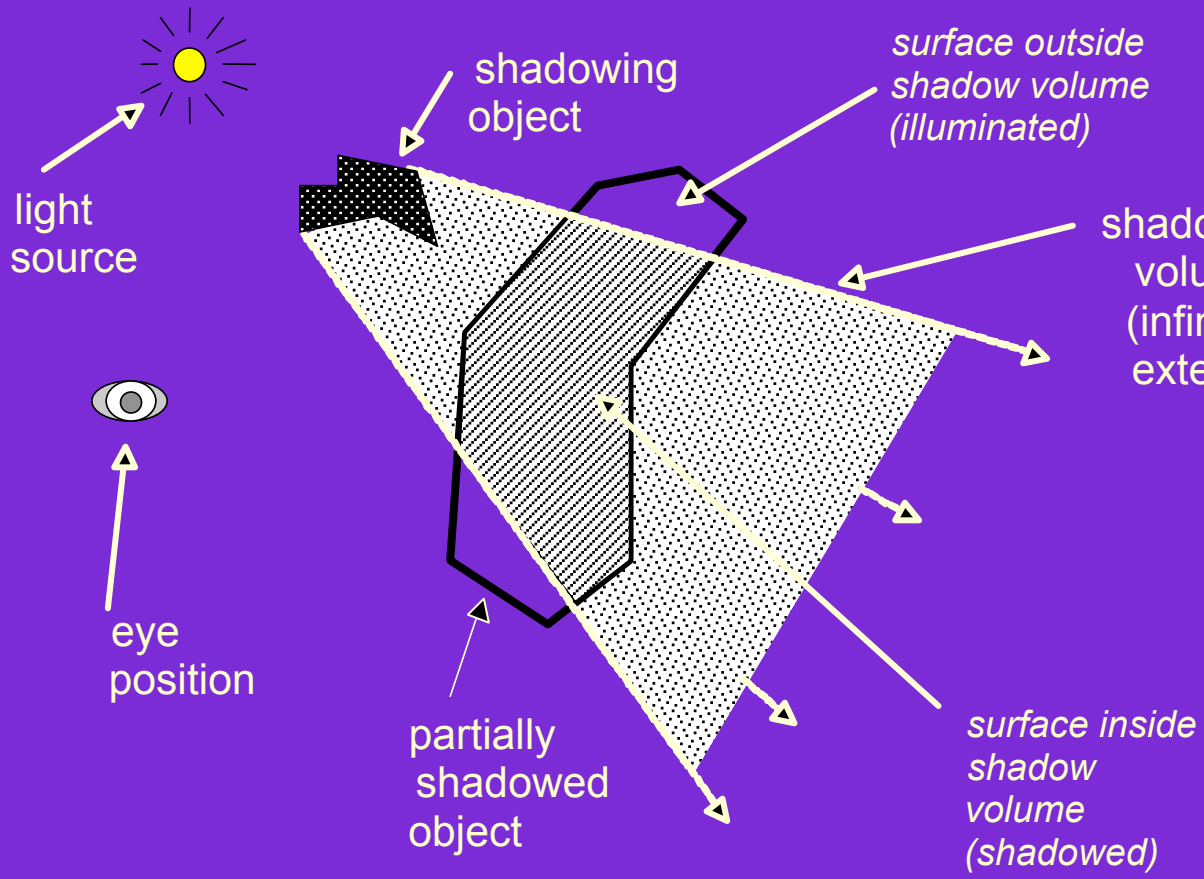
Triangle blocking light source.

"Shadow" volume projected by triangle from the light source.

Use stencil to tag whether pixel is inside or outside of the shadow volume.

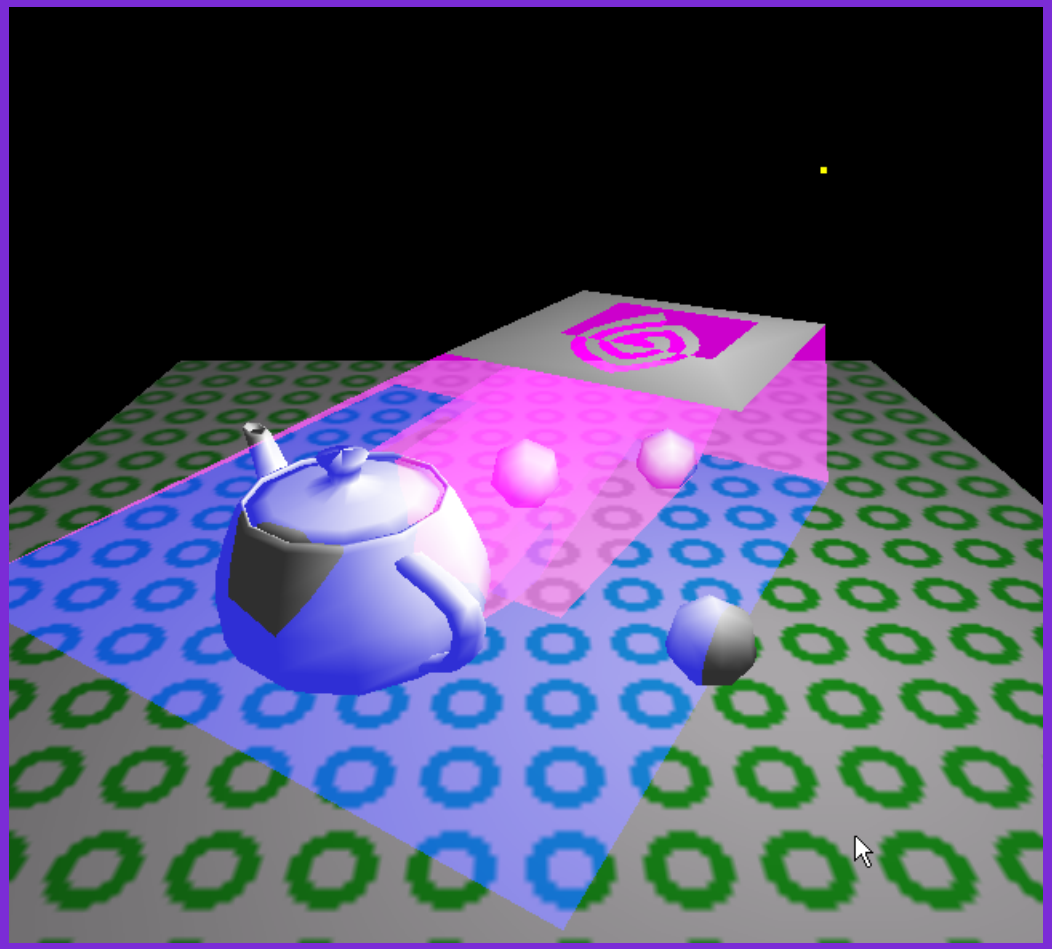
Two passes: light pixels outside volume; no lighting for pixels inside the volume.

2D Cut-away Shadow Volume



-

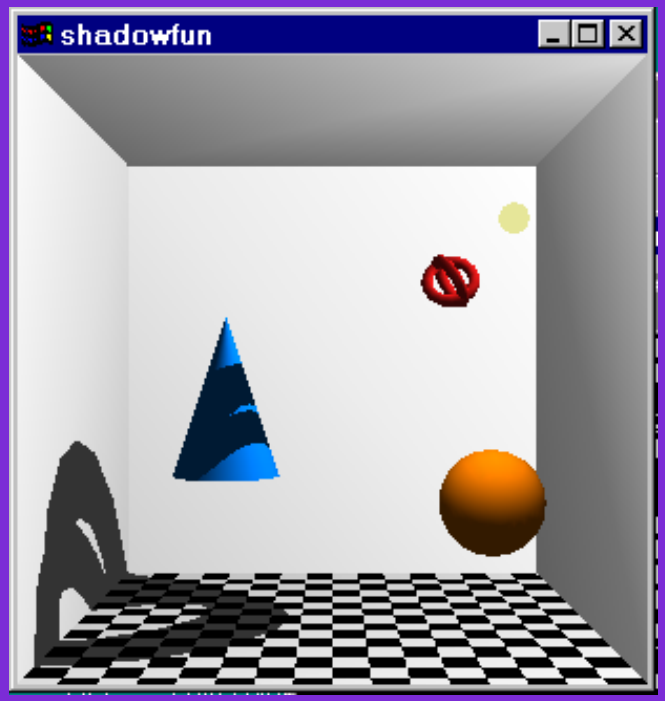
A Complex Shadow Volume

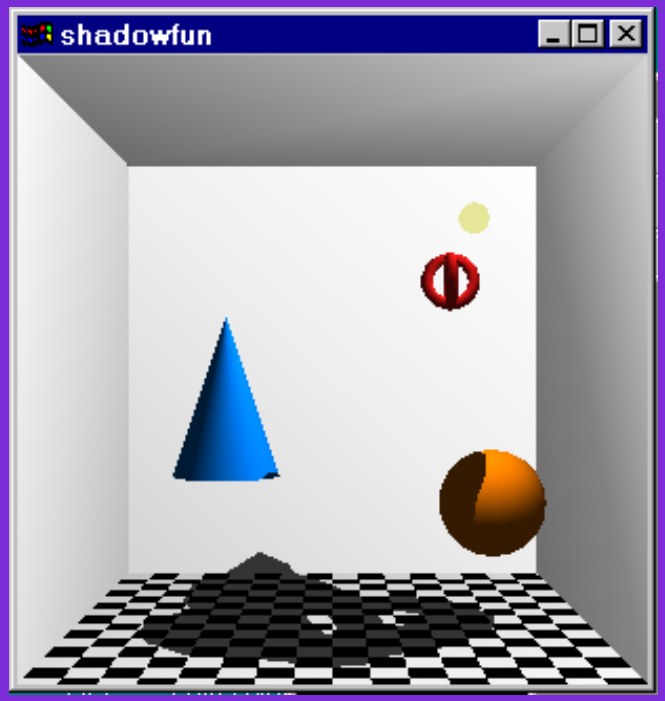


Calculating Object's Silhouette

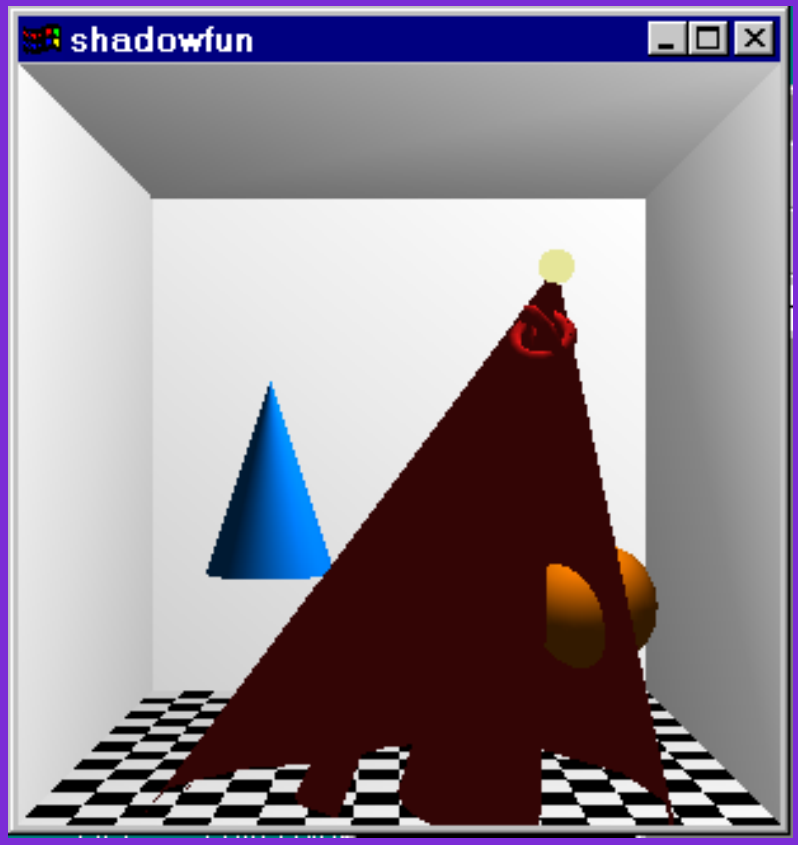
- Single triangle has a simple silhouette to extend a shadow volume from
- Complex objects require a silhouette calculation
  - OpenGL's feedback mechanism and the GLU 1.2 tessellator can help you
- Dynamic shadow volumes are cool!

Complex Shadows On Objects





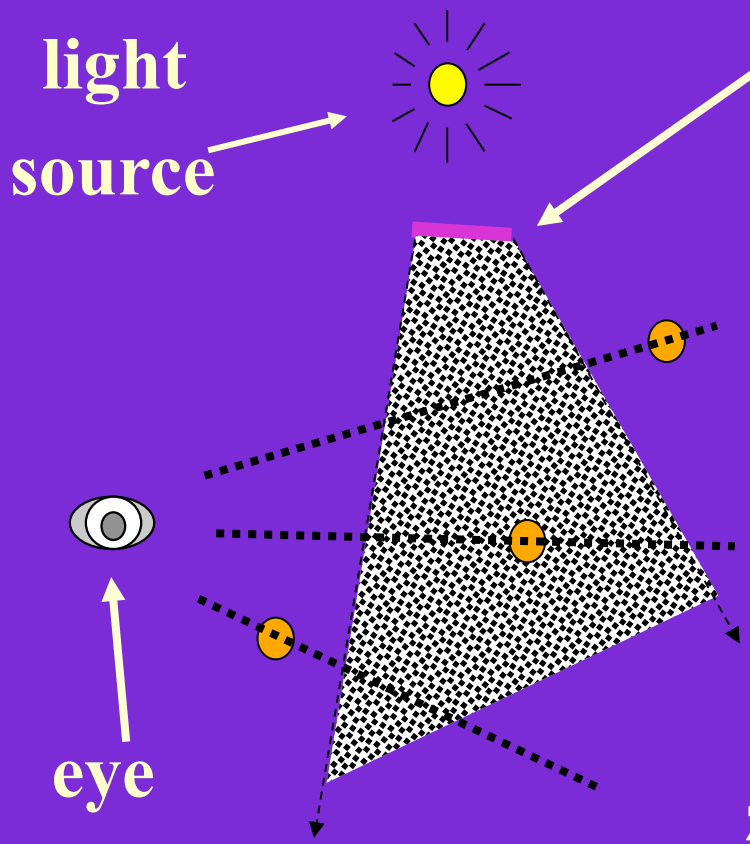
The Complex Shadow Volume



Stencil Shadow Algorithm

- Render scene's depth values into the depth buffer
- Invert stencil bits when depth test passes when rendering the stencil volume
  - glStencilOp(GL_KEEP, GL_KEEP, GL_INVERT);
- Pixels outside the shadow volume are inverted an even number of times
- Pixels inside the shadow volume are inverted an odd number of times

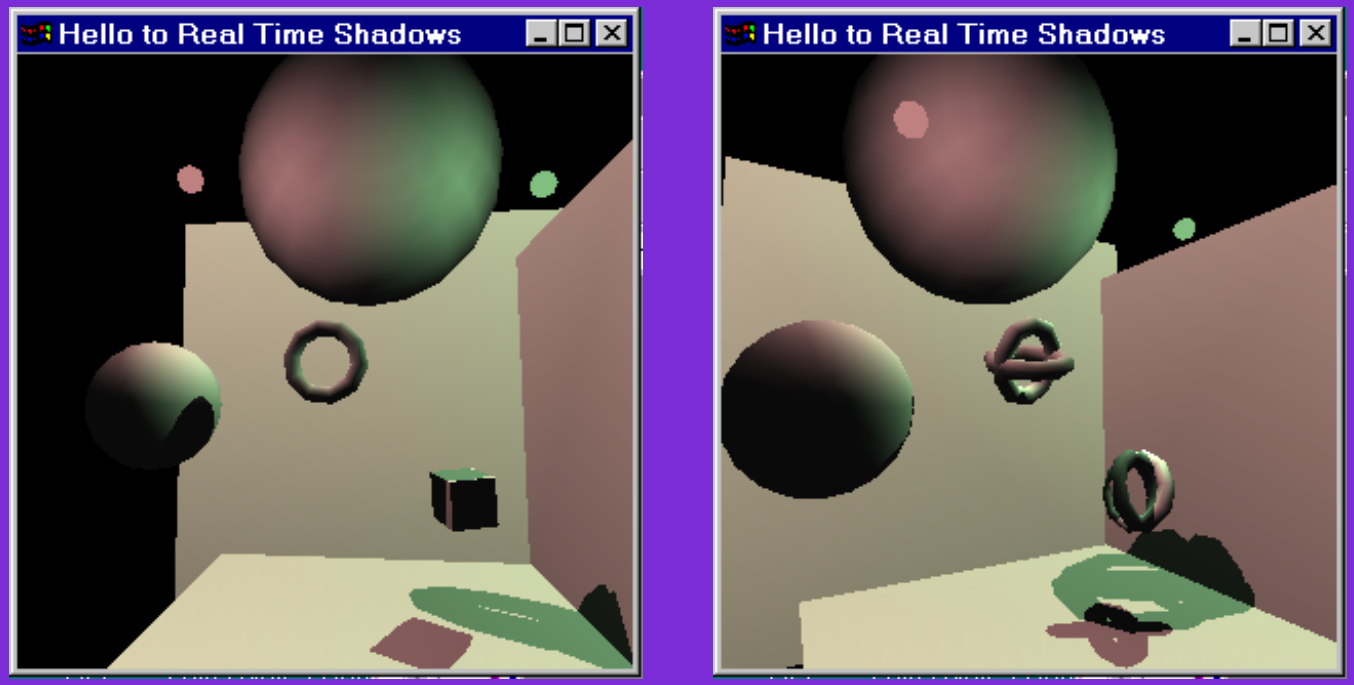
Inverting Examples



Render In and Out of Shadow

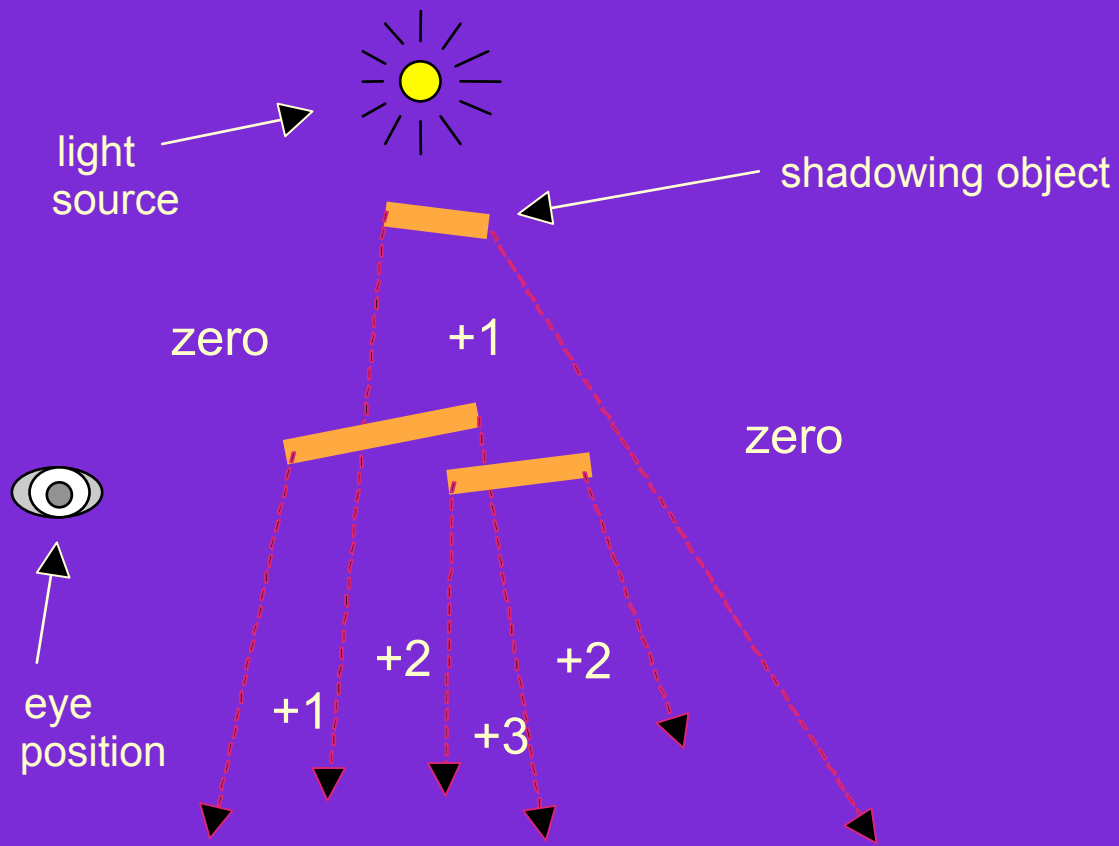
- Render the scene with the light disabled, update only pixels with an odd stencil bit setting
- Render the scene with the light enabled, update only pixels with an even stencil bit setting
- Shadows get rendered!

Multiple Objects and Lights



Notice shadows from different light sources overlap correctly.

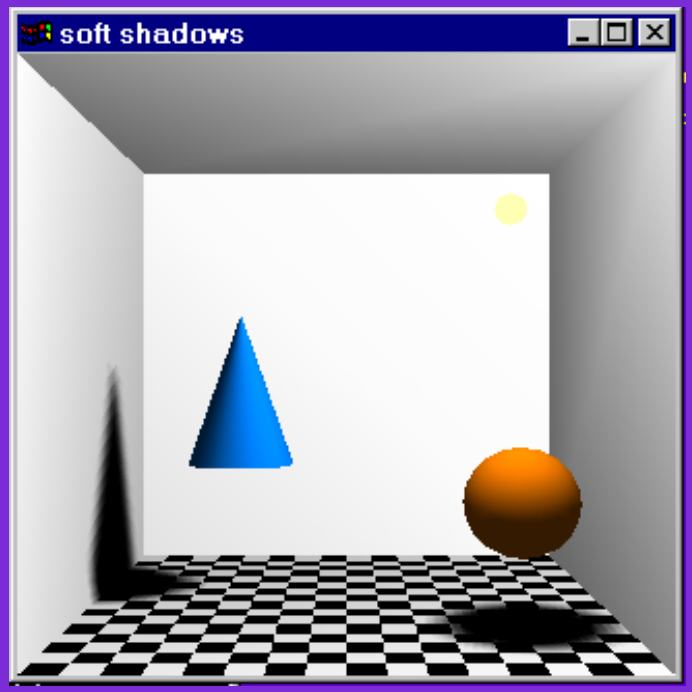
Counting Enters & Leaves



Stencil Counting

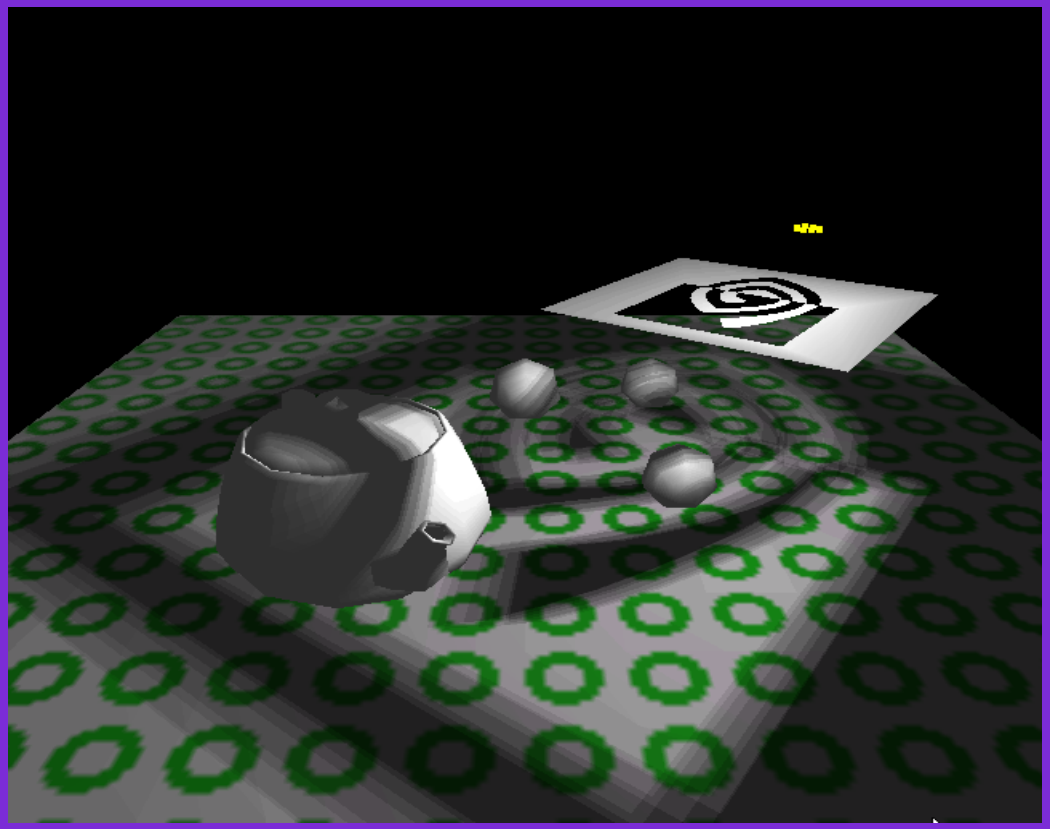
- Use the GL_INCR stencil operation for front facing shadow volume polygons
- Use the GL_DECR stencil operation for back facing shadow volume polygons
- Takes two passes
- Can trivially extend shadow volume from every polygons
- Watch out for cracks!
- Consumes tons of pixel fill rate

Jittered Shadow Volumes



Multiple passes with shadow volumes and the light source slightly jiggled between passes. Accumulation buffer sums up the passes.

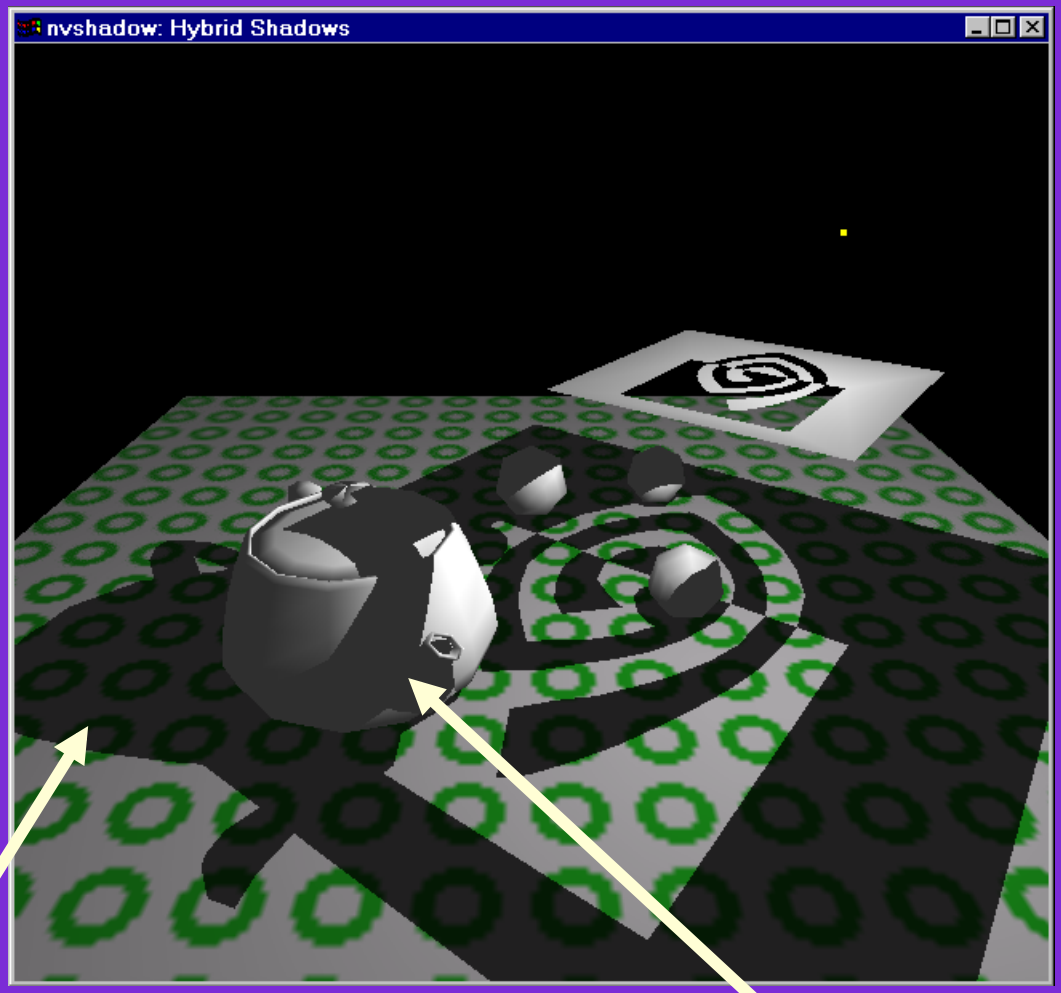
Soft Shadows



Careful about banding artifacts.

Slow: linear cost per shadow volume sample.

Hybrid Shadows

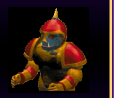


Planar projected shadows

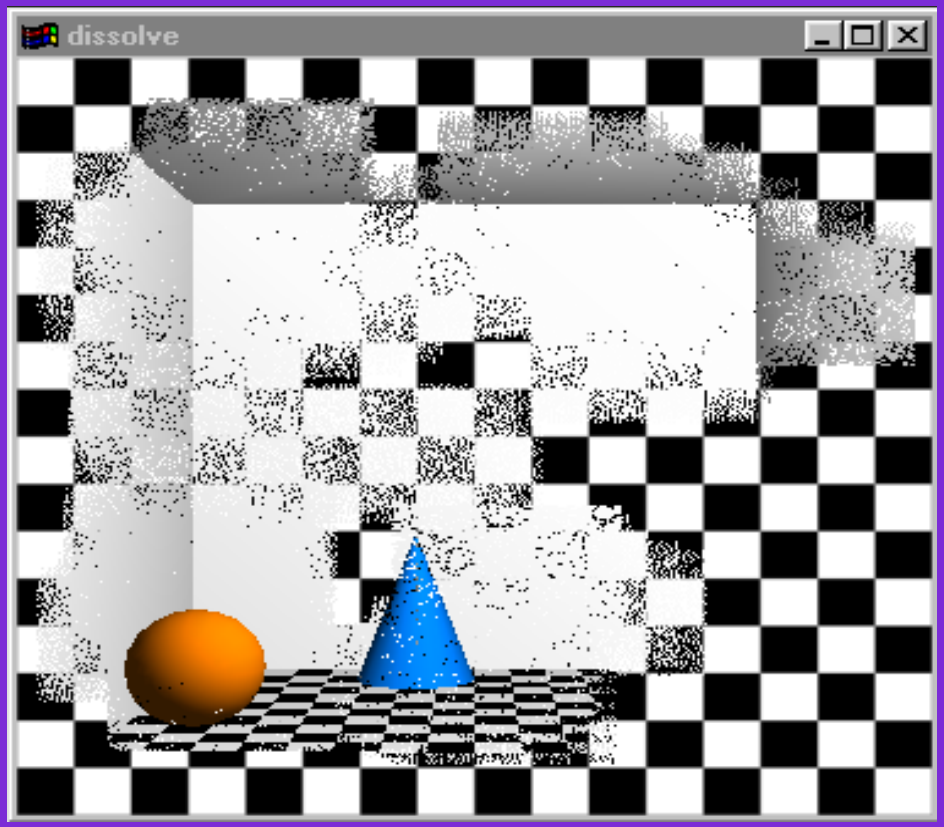
Shadow volumes

Other Stencil Uses

- Digital dissolve effects
- Handling co-planar geometry such as decals
- Measuring depth complexity
- Constructive Solid Geometry (CSG)



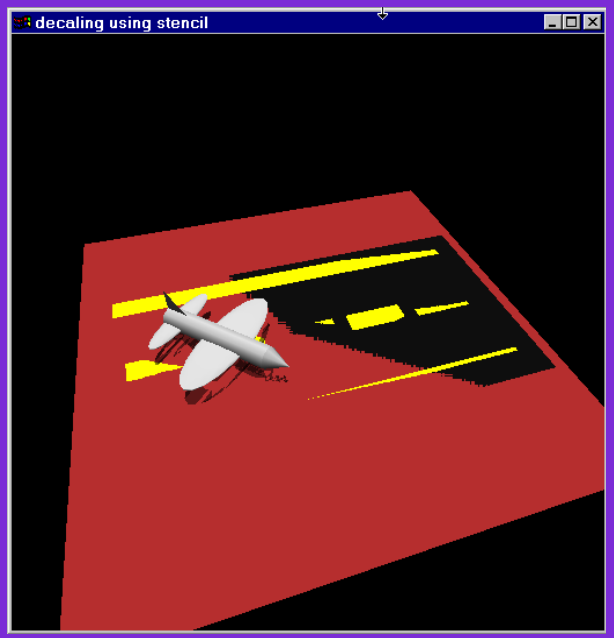
Digital Dissolve

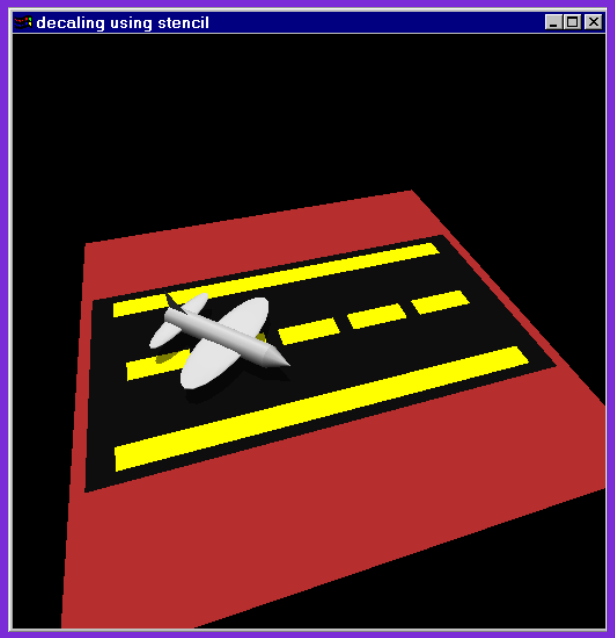


Stencil buffer holds dissolve pattern.

Stencil test two scenes against the pattern

Co-planar Geometry





Shows "Z fighting" of co-planar geometry

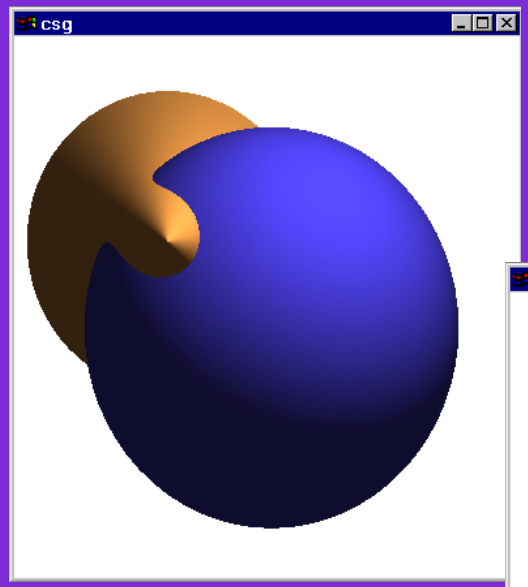
Stencil testing fixes "Z fighting"

Visualizing Depth Complexity

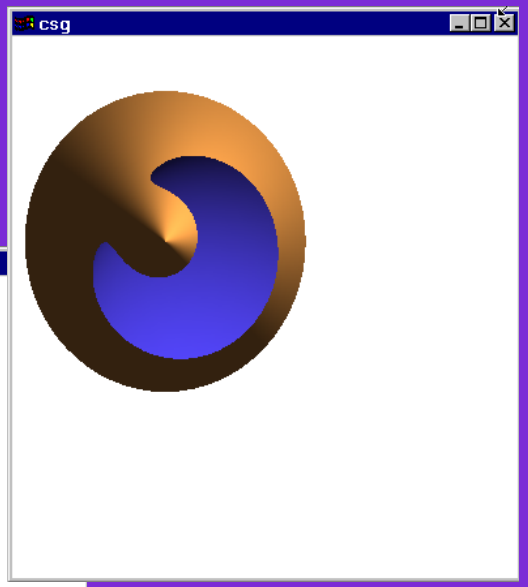
Use stencil to count pixel updates, then color code results.

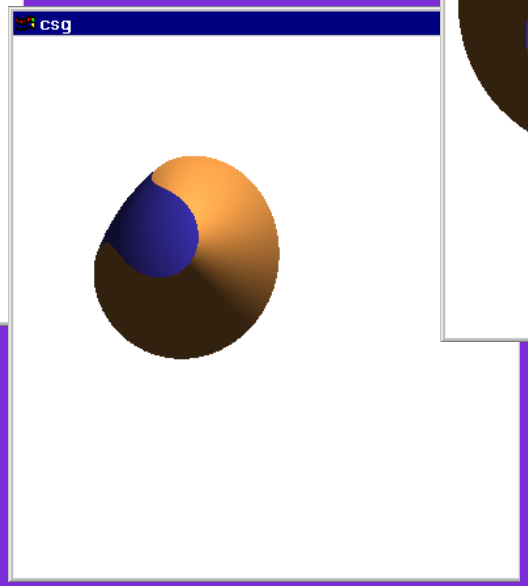
Constructive Solid Geometry

cone UNION sphere



sphere MINUS cone





cone INTERSECT sphere

Conclusions

- Stencil testing = improved visual quality, unique effects
- Games using stenciling today
  - Quake 3 - awesome volumetric shadows
  - Unreal - wicked shadow effects
- Christmas 1999 will have best cards supporting 32-bit rendering with stencil
- Assume 8/24 stencil/depth hardware when depth testing gives you stencil “for free”