6800 LEAGUES UNDER THE SEA

NVIDIA®

Shadow Considerations

Ashu Rege

Developer Technology Group

Shadows

- One of the most important graphical parts of game engine
- Influence on several aspects of game
  - Artwork creation and pipeline
  - Min spec, fallbacks
  - Shader complexity
  - Batch size
  - Performance

Strategic Considerations

- What objects cast shadows?
- What objects receive shadows?
- How do shadows integrate with the art pipeline?
- What technique for shadows
  - One technique or multiple?
- Static lighting v. Dynamic lighting

Tactical Considerations

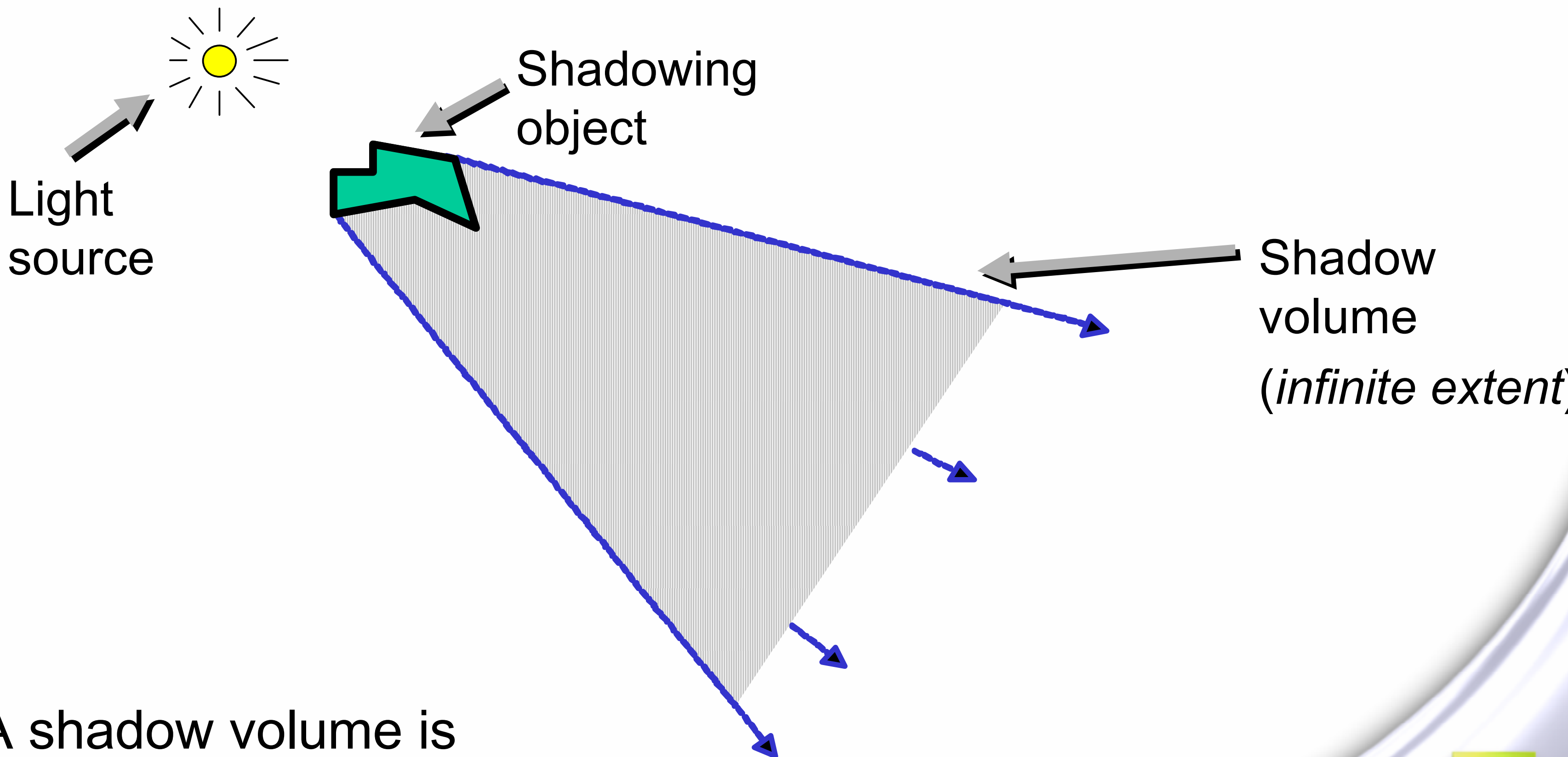
- Light Maps, Precomputed Radiance Transfer, Blobs, ...
- Shadow Volumes or Shadow Maps?
  - Both?
- Issues arising from usage of either
  - World Geometry v. Local Geometry
  - Aliasing problems
  - CPU side computations v. GPU computations
  - ...

NVIDIA.

Two Broad Approaches

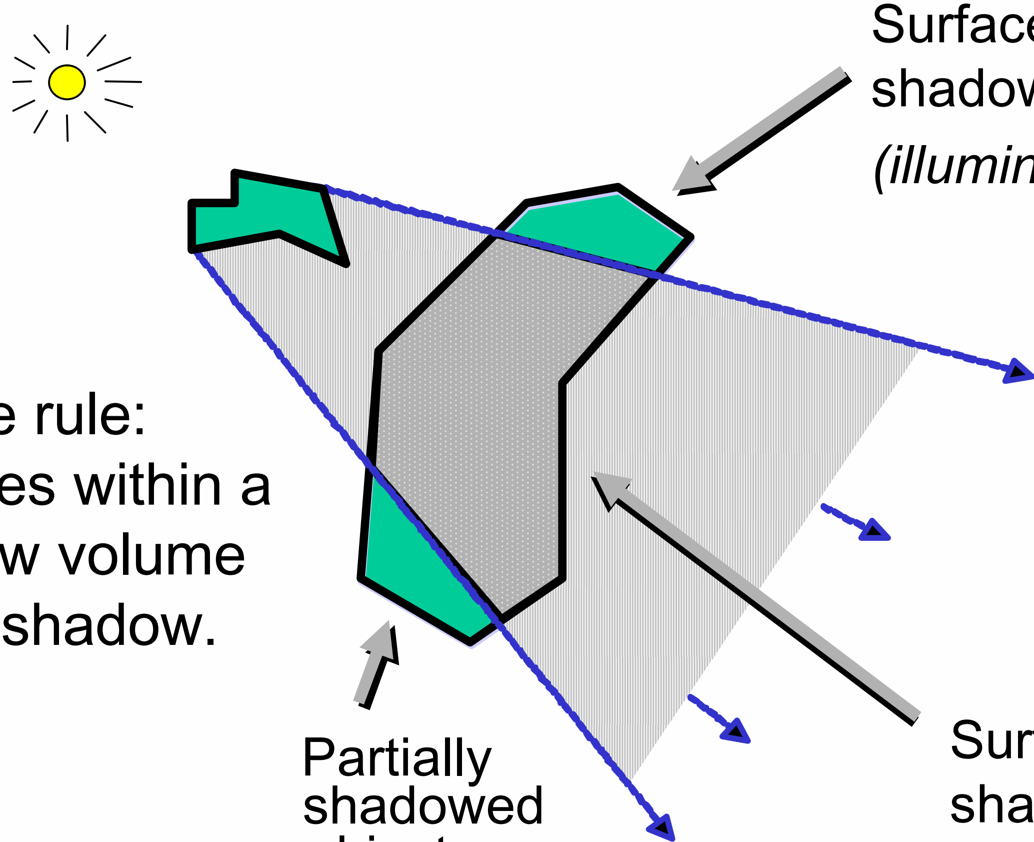
- *Shadow Volumes* and *Shadow Maps*
- No one ‘right’ technique
- Shadow volumes
  - Mathematically elegant, ‘complete’, omni-directional
- Long term, however, we expect shadow maps to be more widely used
  - Better scaling with GPU power
  - Softer edges
  - Applicable to different kinds of geometry
    - No alpha test issues

Shadow Volumes – Basic Concept



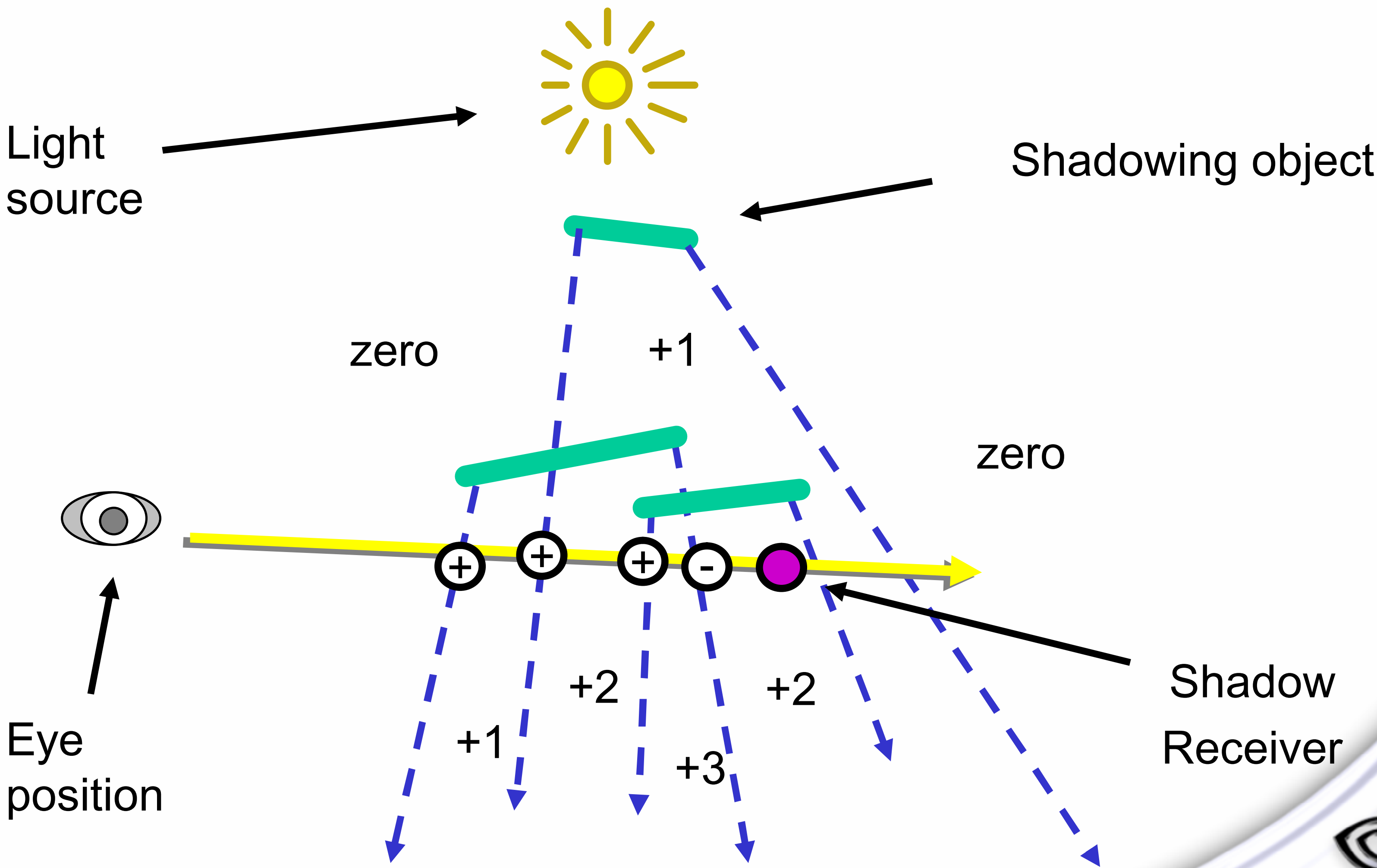
A shadow volume is
simply the half-space defined
by a light source and a shadowing object.

Shadow Volumes – Basic Concept (2)



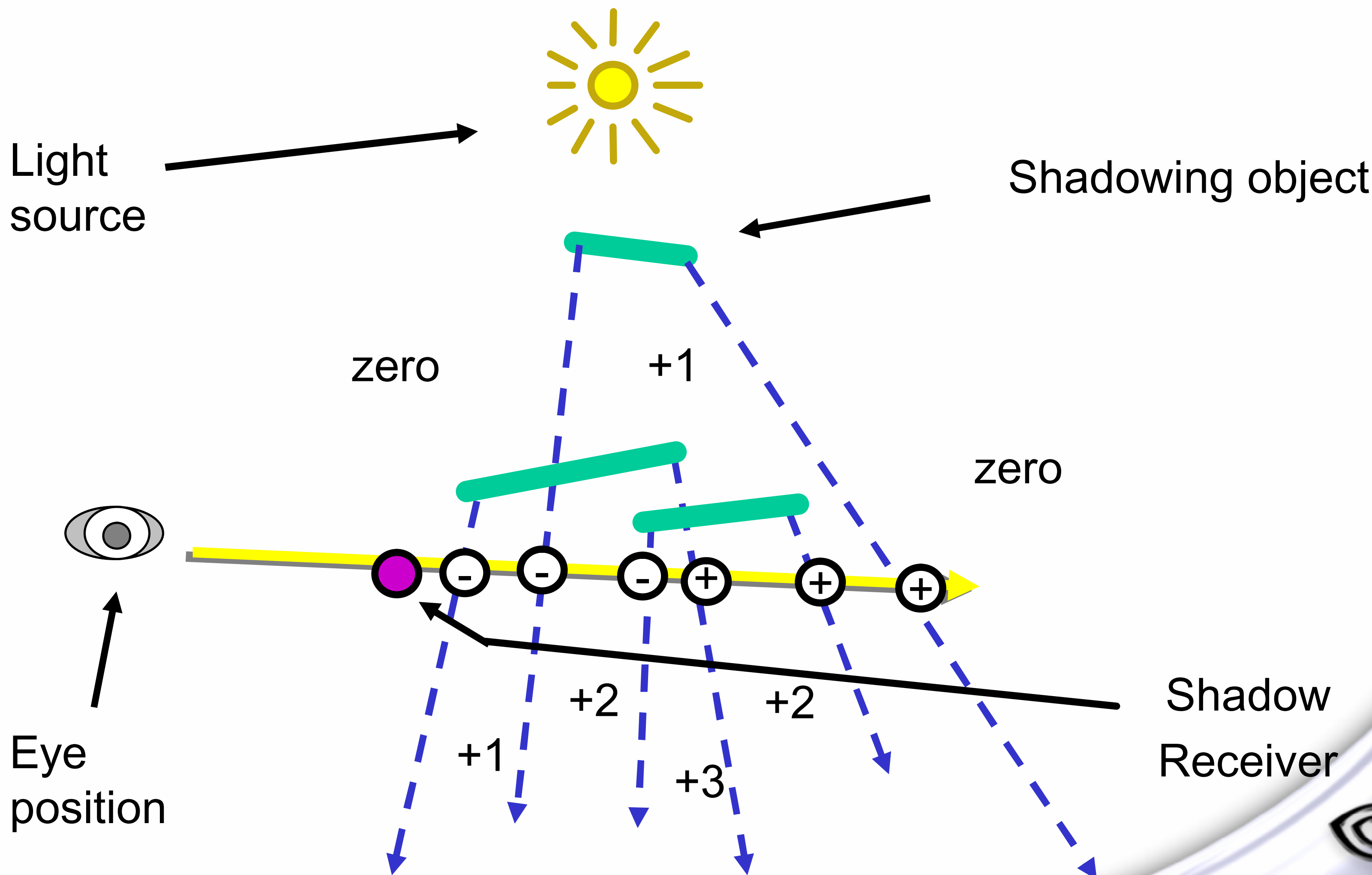
Simple rule:
samples within a shadow volume are in shadow.

Stencil Shadow Volumes (zpass)



Shadow Volume Count = +1+1+1-1 = 2

Stencil Shadow Volumes (zfail)



Shadow Volume Count = -1-1-1+1+1+1 = 0

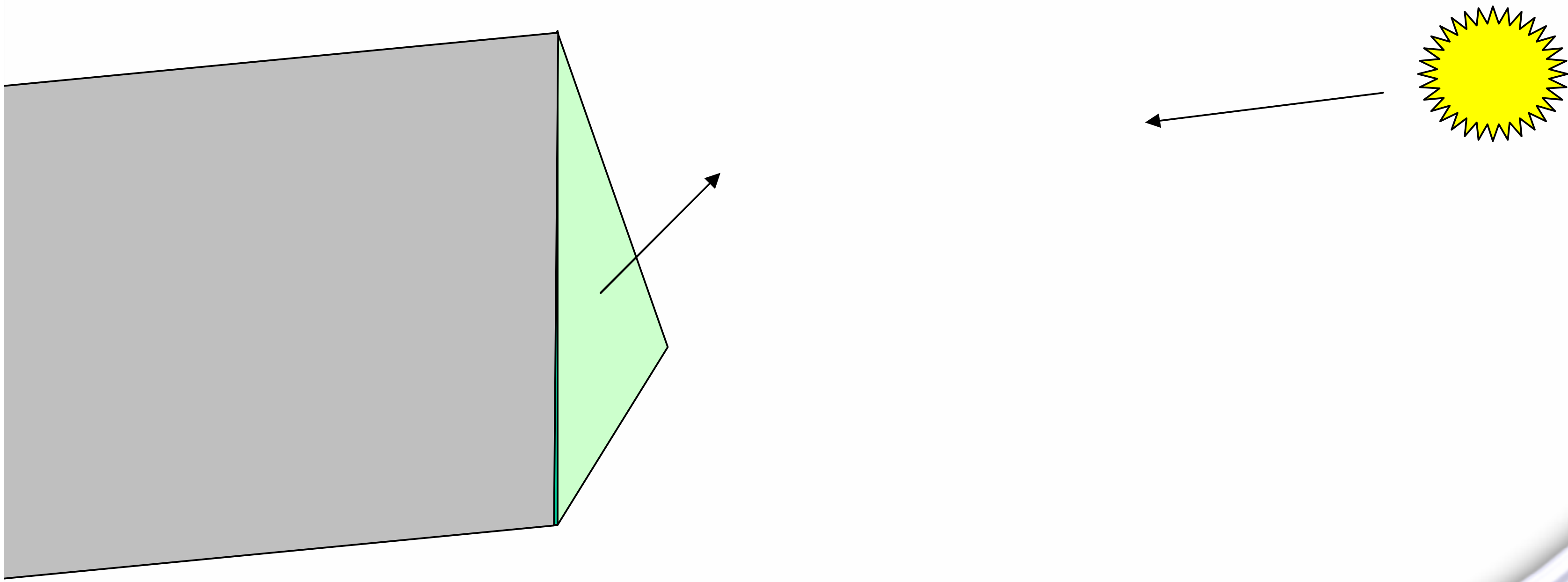
Shadow Volumes – Silhouettes

- How to compute volumes?
- Compute (projected 2D) silhouettes instead and extrude
- One big question to answer when using shadow volumes is how to determine silhouettes
  - On CPU, performing edge tests
  - On GPU, using degenerate geometry on each edge

Silhouette Computation on the CPU

- Requires faces to know neighboring faces
  - For each face
    - Calculate dot product of face normal with light vector
  - For each face
    - Check 3 neighboring faces' dot products
    - If dot product of face a is <= 0.0, and face b is > 0.0
      - Then the edge between a & b is a silhouette edge
    - Construct quad along edge by extruding away from light

CPU Silhouettes – Quad Extrusion



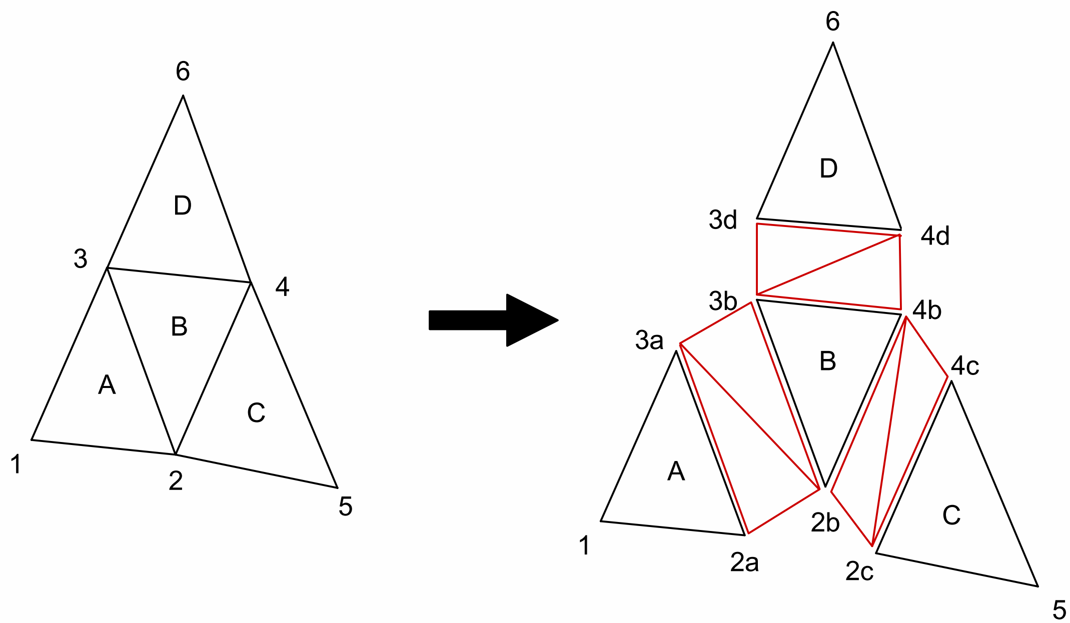
Pros and Cons of CPU Silhouettes

- + Straightforward algorithm
- + Linear in the number of faces
- + Only need to recompute when light or objects move (relative to each other)
- + Works well with skinning
  - Skin on CPU, then compute silhouette
- - Can be expensive for dense meshes

Shadow Volumes on the GPU

- Insert ‘degenerate’ quads at each edge of mesh
- Each vertex in the quad has
  - a position
  - a copy of the face normal
  - an extrusion factor of 0 or 1
- For 2 of the quad’s vertices
  - The extrusion factor is 0
  - For the other 2, the factor is 1
- If the face normal dot the light direction is zero, extrude the vertex away from the light

Volumes on the GPU – Bloating



Original triangle mesh
6 vertexes
4 triangles

Bloated triangle mesh
12 vertexes
10 triangles

Formula for geometry:

$v_{bloat} = 3 * t_{orig}$

$t_{bloat} = t_{orig} + 2 * e_{orig}$

Bloated geometry based only on number of *triangles* and *edges* of original geometry.

A <u>*lot*</u> of extra geometry!

Skinning With GPU Extrusion

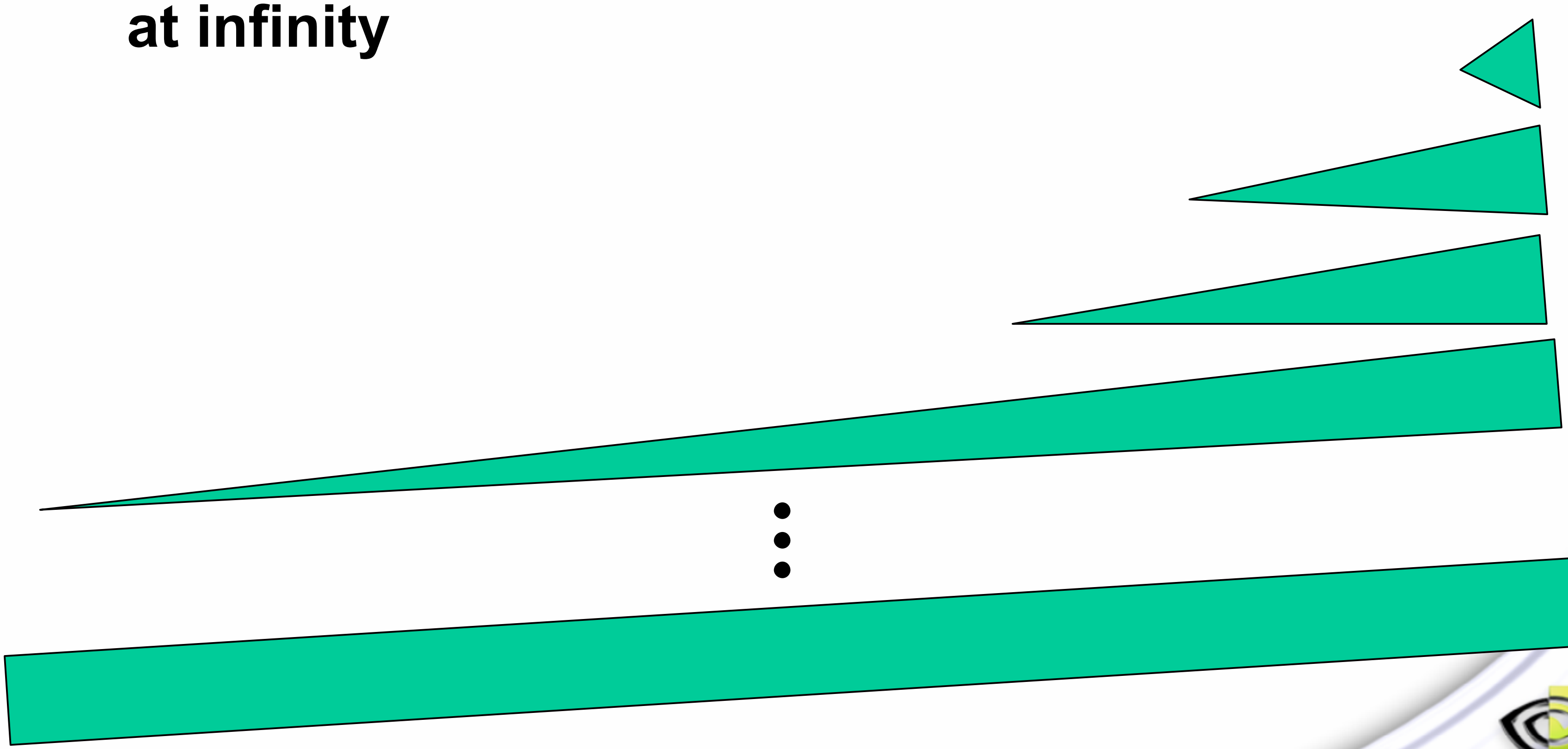
- If performing a non-linear transformation, like skinning, you don’t know the face normal
  - Unless you know all 3 of the face vertices’ positions
- So, if doing skinning, you must, for each edge of the model
  - Store all 3 vertex positions making up this face
  - Perform skinning on each
  - Then test the face normal, & extrude
- Very expensive for skinned models

Good To Be GPU Bound, Right?

- Depends: vertex bound, pixel bound, or setup bound?
- Current generation hardware: pixel shader horsepower has grown much faster than other two
- Setup in particular is still 1-2 clocks per triangle
  - Degenerate triangles eat up setup time
  - Setup bound → Rendering will scale with clock only
  - Clocks haven't gone up quite as much
- Future hardware and API could change this picture

Reducing Setup Dependency

- Turn extruded quads into extruded tris
- A quad can be viewed as a triangle with one vertex at infinity



Quad → Tri

- **Rather than drawing a quad for each triangle edge, draw a triangle with one vertex having a w coordinate of zero for directional lights**
  - **This is known as an external vertex**
  - **Twice as fast if you are setup bound**
  - **One triangle instead of two for a quad**
  - **25% faster if you are vertex bound**
  - **Also has more subtle benefits to rasterizer, b/c the quad isn't two skinny triangles, but one long, fat triangle**

Other Optimizations For SSVs

- Two-sided Stencil (DX9)
  - Send both front and back faces at same time
- Semi-automatic shadow volume extrusion
  - CPU performs possible silhouette edge detection for each light
  - GPU projects out quads from single set of vertex data based on light position parameter
  - Doom3's approach
- Depth bounds, depth clamping
- See Everitt and Kilgard presentations/papers for all things SSV (www.developer.nvidia.com)

Pros and Cons of SSVs

- + Automatic self-shadowing
- + Omni-directional lights
- + Minimal aliasing and resolution issues
- - No area lights, no soft shadows
- - Mesh must be 2-manifold (closed) w/ connectivity
- - Consumes fill rate
- - Need silhouette computation
  - Could eat preciouss CPU cycles
- - Not compatible with alpha test
- - Inherently multi-pass!
- - Popping esp. with low poly counts

Pixel Power!

- Going forward, pixel shader math horsepower will grow faster than :
  - Texture fetching & filtering
  - Vertex shader horsepower
  - Triangle Setup
  - CPU power
  - Memory bandwidth
  - Just about anything else

Leveraging Pixel Power For Shadows

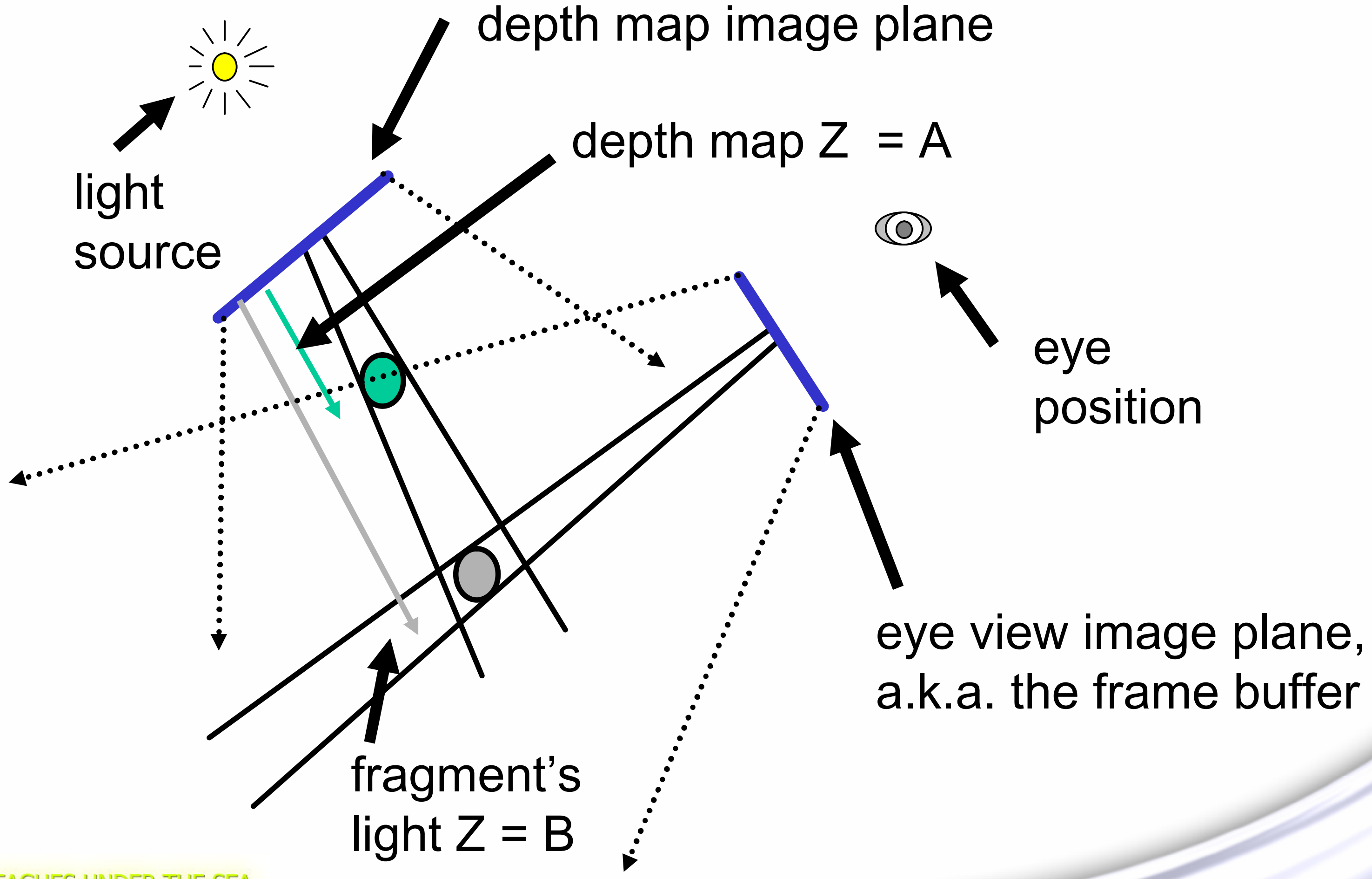
- Shadow Maps
- Image-space technique
  - No knowledge of scene geometry
  - But aliasing…
- Well-known technique
  - Ubiquitous in production Renderman shaders
- Hardware-accelerated since GeForce3
- Scales with *pixel* power

Shadow Maps – Basic Algorithm

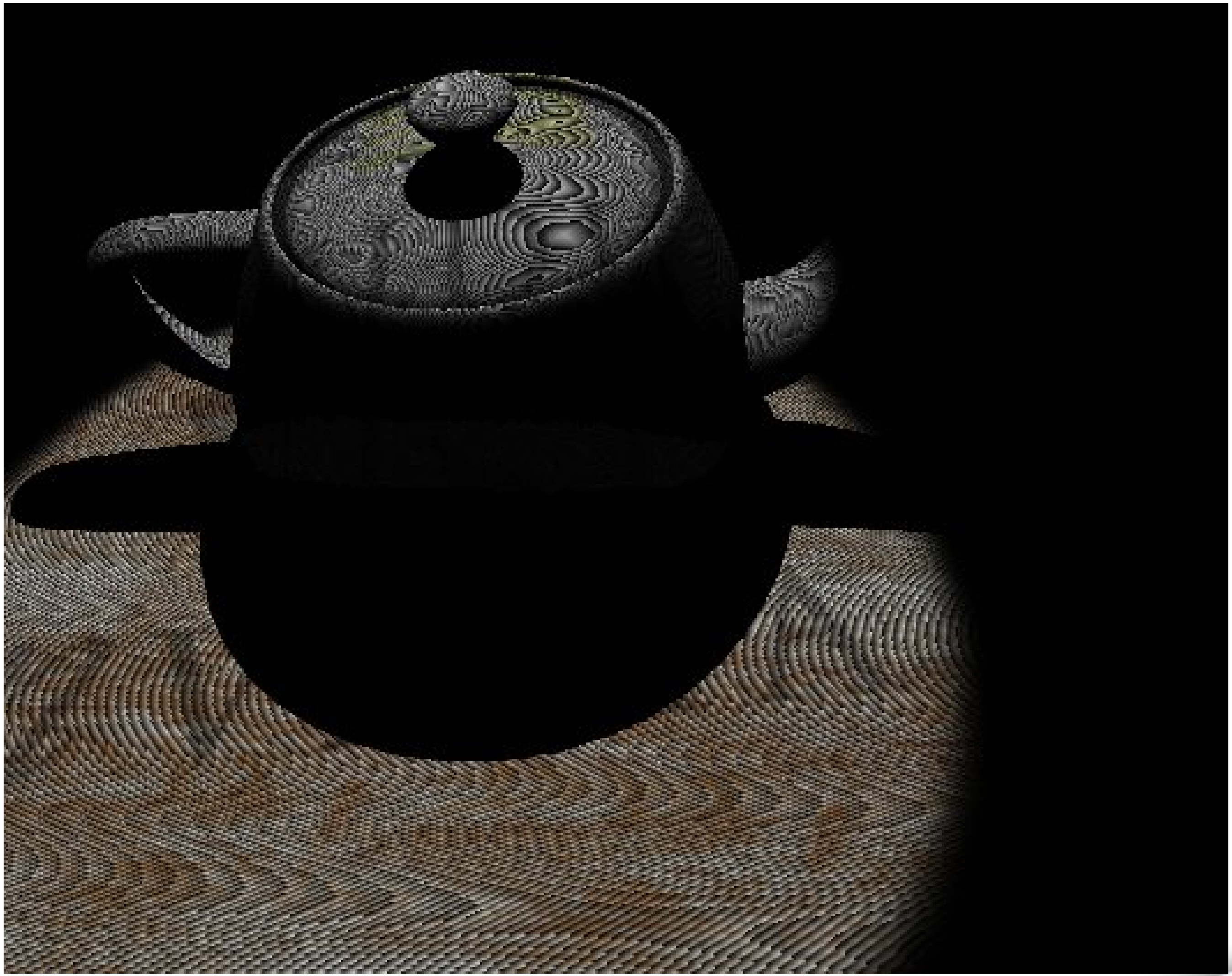
- **Several variations on the same theme**
- **Light can "see" point ⇔ Point is not in shadow**
  - **Render objects from the light's POV, storing *depth* from the light into the shadow map**
  - **Render objects from the camera's POV, but also test their depth with respect to the light**
  - **If this object's depth ~= the closest object in the shadow map, then object is lit**
  - **Else object is in shadow**

Shadow Maps – Example

The A < B shadowed fragment case

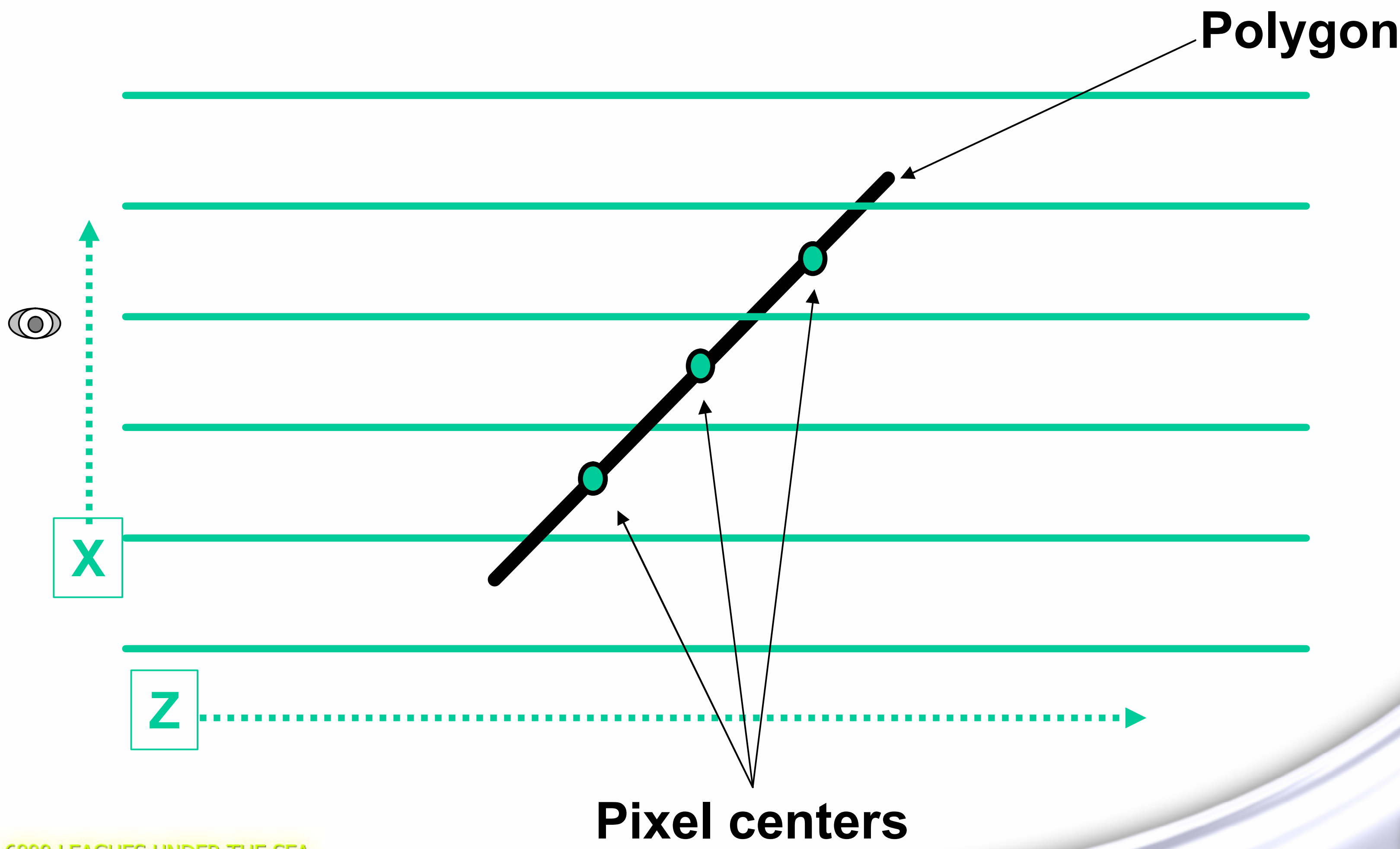


The Result So Far…



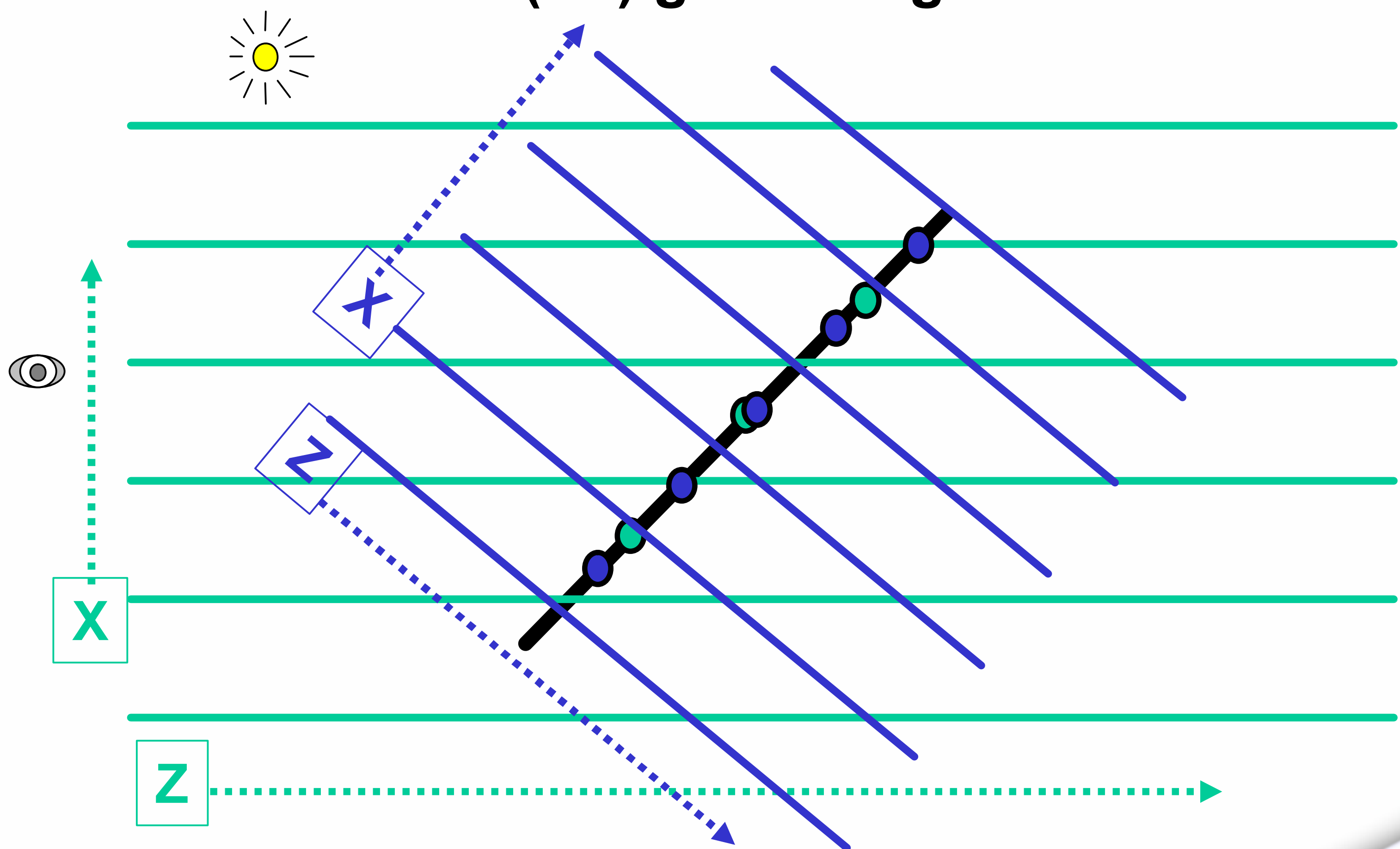
What Is Going On?

- Consider 2D view of polygon (x and z == depth)



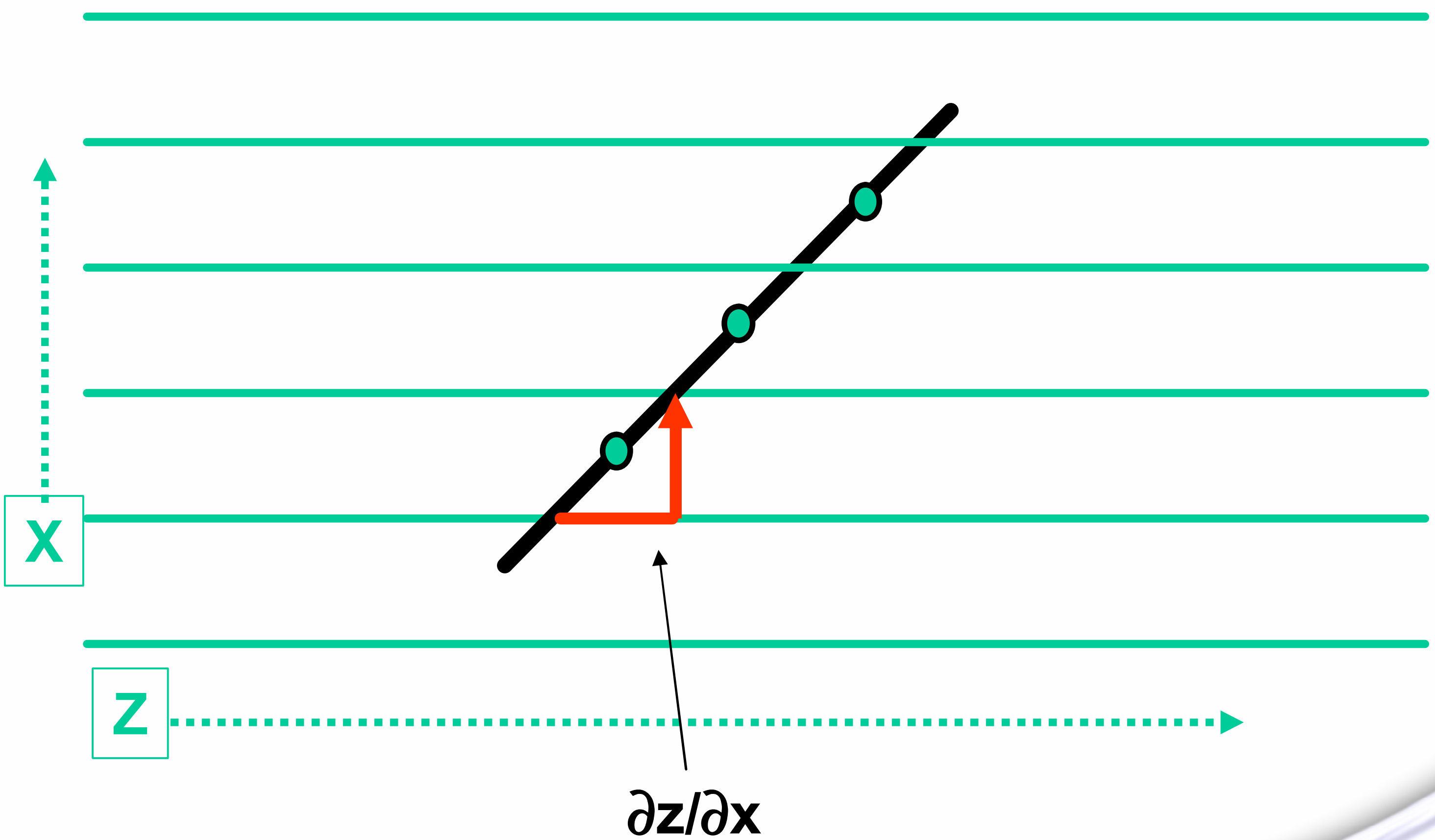
Depth Aliasing

- Add another (2D) grid for light view



Depth Aliasing – Measuring Error

- **Change of Z w.r.t. X**



Depth Aliasing – Maximum Error

- **Pixel center is re-sampled to shadow map grid**
- **The re-sampled depth could be off by**
  $\pm 0.5\ \partial z/\partial x$ and $\pm 0.5\ \partial z/\partial y$
- **The maximum absolute error would be**
  $|\ 0.5\ \partial z/\partial x\ | + |\ 0.5\ \partial z/\partial y\ | \approx \max(\ |\ \partial z/\partial x\ |\ ,\ |\ \partial z/\partial y\ |\ )$
  - **Assumes the two grids have pixel footprint area ratios of 1.0**
  - **Otherwise *relative resolutions* of grids will determine scale**

Simple Bias Will Not Work

- Post-perspective divide → depth distribution is non-linear
- Need to bias in post-projective space
- Need to account for slope of polygon

Depth Bias

- DX9:

  Offset = m * D3DRS_SLOPESCALEDEPTHBIAS + D3DRS_DEPTHBIAS

  - Where m = max(| ∂z/∂x | , | ∂z/∂y |)
- Offset is added *before* the depth test and *before* depth value is written into shadow map
- Exactly what we want!
  - Set *slope scale bias* to adjust for resolution scale
  - Set *depth bias* to adjust for total error
- (OpenGL: *glPolygonOffset* is similar)

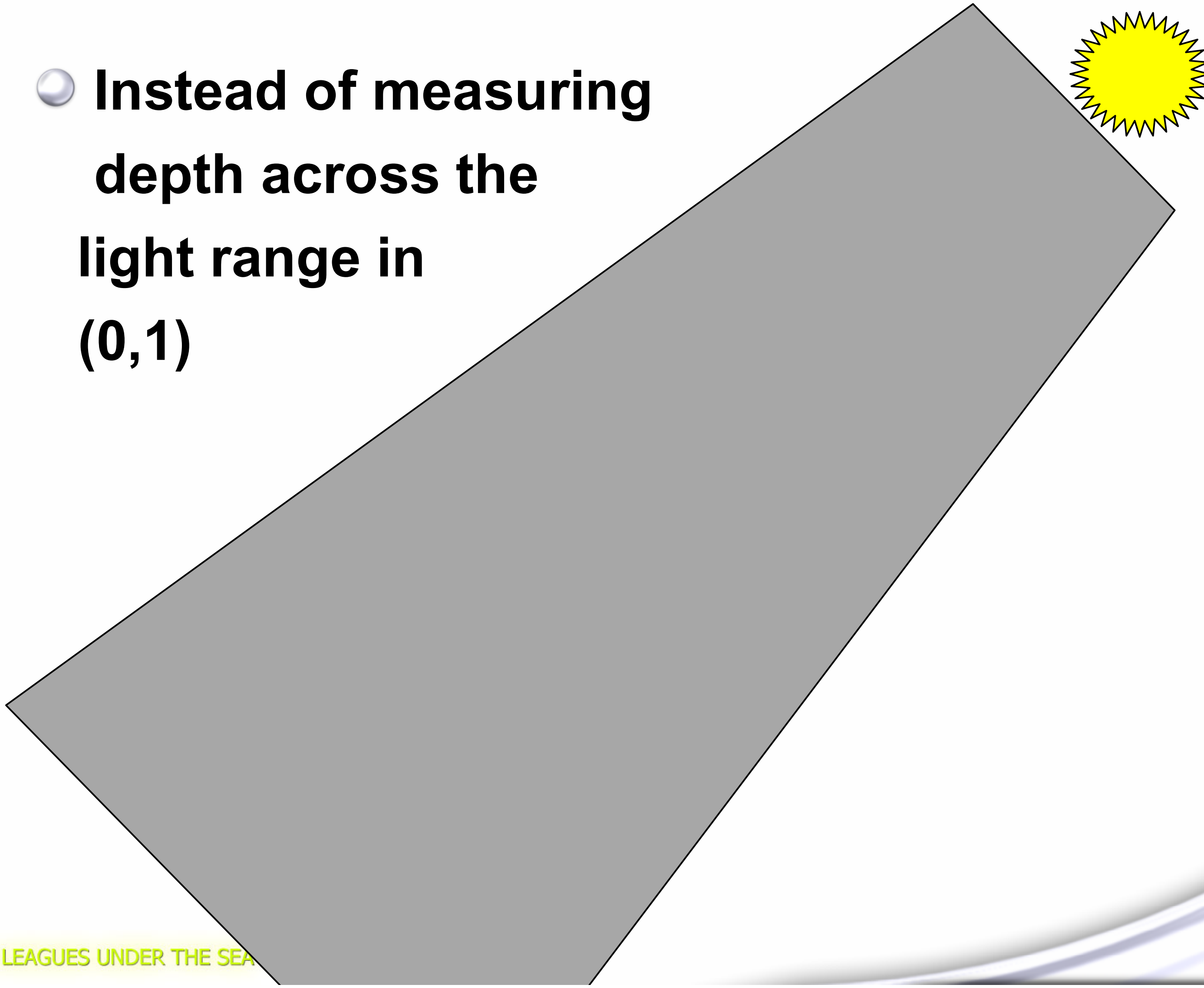
Are We Done?

- Unfortunately, not quite
- How to select bias
  - Magnified shadow maps require larger scale
- Problem: *depth precision* (or lack thereof)
  - Use higher precision depth: D16 → D24
  - Not a scalable solution
- Problem: *perspective aliasing*
  - Depth distribution is not uniform
  - Objects distant from light may be close to viewer
  - Shadow texels near camera can be very large
  - Use higher res → again not scalable

nVIDIA.

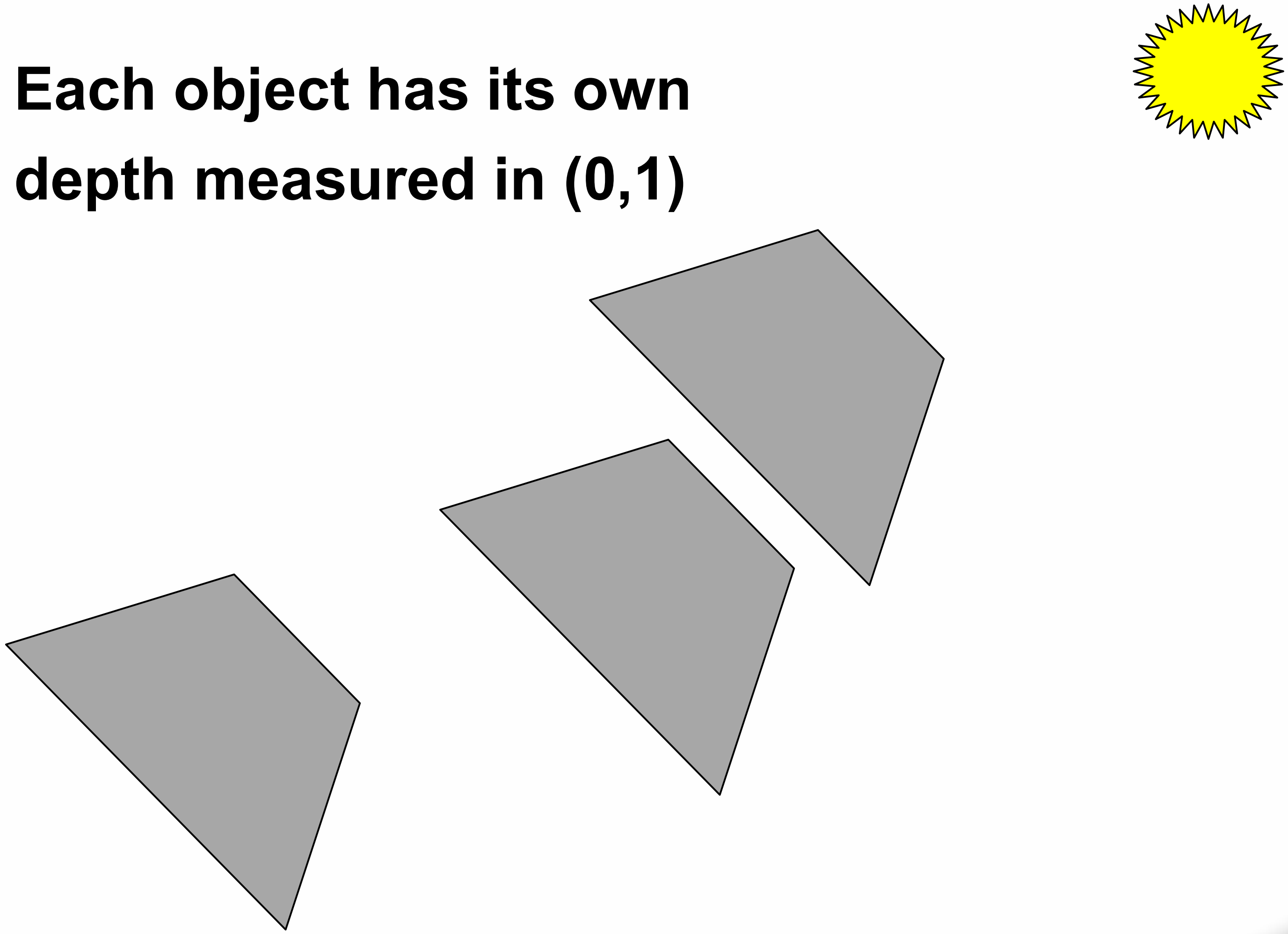
Per-Object Shadow Maps

- **Instead of measuring depth across the light range in (0,1)**



Per-Object Shadow Maps

Each object has its own depth measured in (0,1)



Per-Object Maps – Pros and Cons

- Increased depth precision per object
- Possible reuse per frame
- Can pack multiple shadow maps into 'shadow map atlas'
  - Saves render target switches
- Could get away with 8 bits of depth
  - Support self-shadowing in ps1.1 hardware

- Only supports local objects, not world geometry
- Too many casters → performance problems
  - Merge close casters into one frustum

What About Perspective Aliasing?

- **Shadow texels far from light, close to viewer get magnified**
  - **Fundamental property of projection transform**
- **Sampling is done independent of the view matrix**

- **Idea: Transform light space in a view-*dependent* manner**

Perspective Shadow Maps

- Generate the map in *post-projective* space.
  - Originally proposed by Stamminger/Drettakis, 2002
  - Key Improvements/Elaboration: Kozlov, GPU Gems http://developer.nvidia.com/object/gpu_gems_home.html
- For a directional light
  - Take ‘LookAt’ matrix from post-projective light space to view space
  - Compose with scene View*Projection

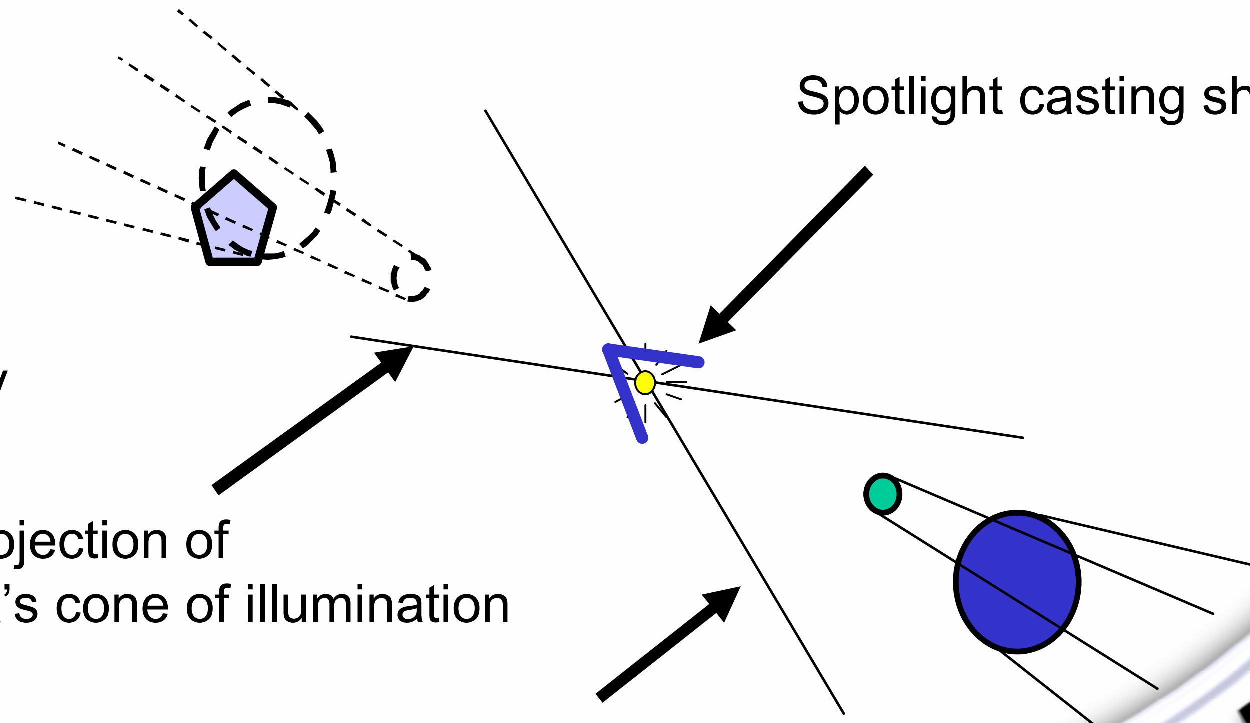
PSMs – Pros And Cons

- Reduces perspective aliasing significantly

- Tricky to implement (and get right)
  - See Gary King's NVSDK demo for implementation
- CPU-side computations needed for speedups
- View dependence →Caching schemes defeated

Are We Out Of The Woods Yet?

- **Just like standard projective textures, shadow maps can back-project**



Eliminating Back Projection

- Modulate shadow map result with lighting result from a single per-vertex spotlight with proper cut off
  - Ensures light is "off" behind the spotlight
- Use small 1D texture – s is planar dist from light
  - Lookup is 0 for negative distances, 1 for positive
- Clip plane positioned at light position OR
- Simply avoid drawing geometry behind light when applying shadow map

Other Tricks With Shadow Maps

- Render *back* faces into map instead of front
  - Leakage moved to less noticeable areas
- Shrink shadow casters
  - Minimize self-shadowing artifacts (works with SSVs)
- Omni-directional shadow ‘cube’ maps (Newhall/King)
  - Simulate cube map with 2D texture
  - Lookup with an auxiliary smaller cube map

Pros and Cons of Shadow Maps

- + Image space → Pixel based
  - Independent of vertex programs – skinning
  - Independent of scene complexity
- + No special requirements for geometry
  - No CPU side computations (in general)
- + Soft shadows, filtering
- + Works great with multi-pass
  - Can collapse multiple lights using SM3.0
  - Compatible with alpha test
- - Omni-directional lights?
- - Resource consumption (textures, render target switching)
- - Aliasing issues

World v. Local Geometry

- Probably best to mix and match techniques
- World Geometry
  - Light maps
  - Stencil Shadow Volumes
  - Precomputed Radiance Transfer
  - Projective Shadow Maps
- Local Geometry a.k.a. ‘objects’
  - Shadow Maps
  - Per-object Shadow Maps
  - Object ID Shadow Maps

Hardware Shadow Maps – Use Them!

- There is no reason not to
- Supported since GeForce3
  - Except GeForce4 MX
- Free Percentage Closest Filtering
  - Weighted average of shadow map comparisons
  - Can combine with higher quality filters
  - Combine with branching in SM3.0 for selective filtering
- Huge perf win v. emulating in shader
- Double speed rendering on GeForce FX and above

Credits and References

- Cass Everitt, Mark Kilgard. Series of presentations and papers on stencil shadow volumes available from developer.nvidia.com
- Sim Dietrich (whose original presentation and ideas I stole)
- Cem Cebenoyan, Gary King (for valuable insights, and posing deep imponderable questions)
- All errors are theirs ☺
- But you can complain to me at: arege@nvidia.com