Advanced Light and Shadow Culling Methods

Eric Lengyel

Terathon Software

GameDevelopers Conference

WHAT'S NEXT GDC:06

Fully Dynamic Environment

- Anything in the world can move
  - Can't precompute any visibility information
- Lights completely dynamic
  - Can't precompute any lighting information
  - Shadows also completely dynamic

Problems to Be Solved at Run-time

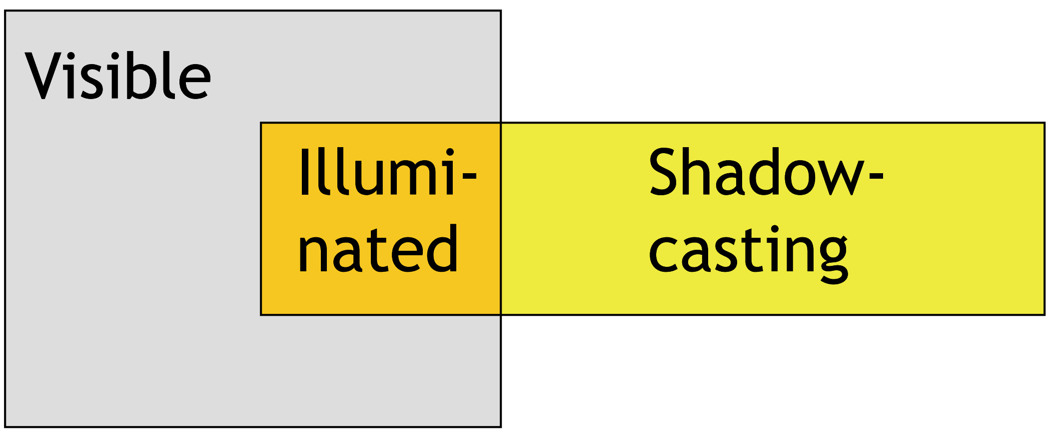
- Determine the set of objects visible to the camera
- Determine the set of lights that can influence any region of space visible to the camera
- For each light, also determine what subset of the visible objects are illuminated by the light

Problems to Be Solved at Run-time

- Determine the set of objects that could possibly cast shadows into the region of space visible to the camera
  - For each light, this is a superset of the set of the illuminated objects that are visible to the camera

Sets of Objects

- Visible set
- Illuminated set (x *n* lights)
- Shadow-casting set (x *n* lights)



GameDevelopers Conference

Visibility Determination

- Organize the world in some way
  - Tree structures (BSP, octree, etc.)
  - Hierarchical bounding volumes
  - Portal system
- A combination of these can work extremely well
- Portals fine outdoors as well

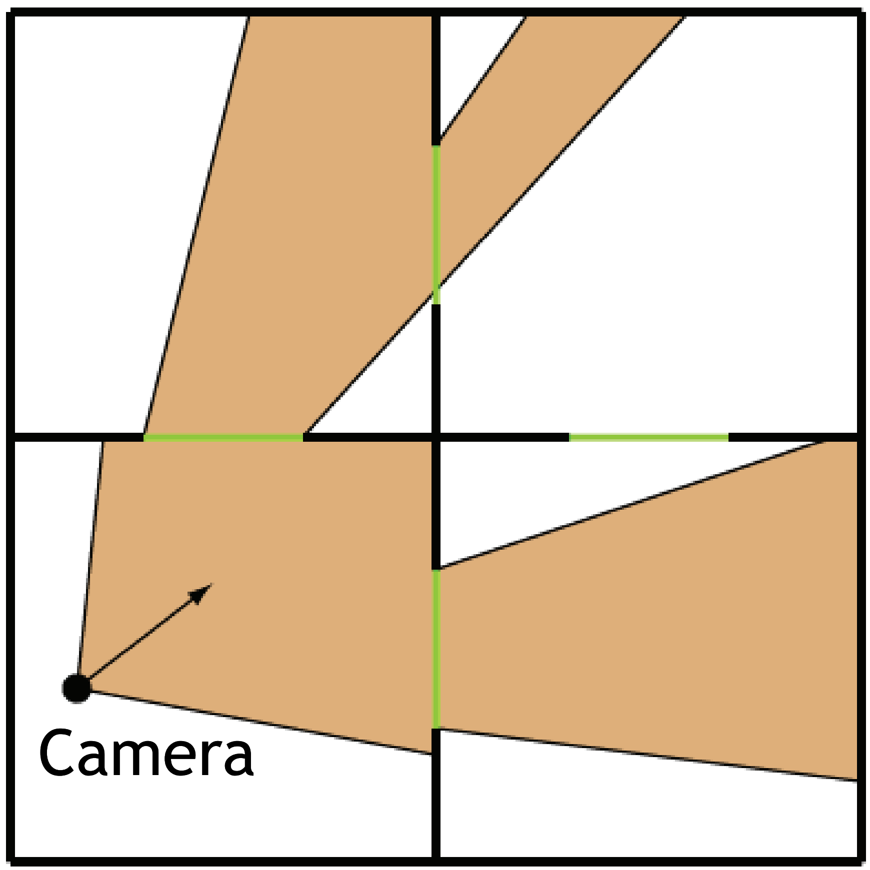
Portal Systems

- World divided into ***zones***
  - A zone is the region of space bounded by a convex polyhedron
- Zones are connected by ***portals***
  - A portal is a planar convex polygon
  - From the front side, a portal's vertices are wound CCW

Portal Systems

- During visibility determination, only have to worry about zones that can be seen through a sequence of portals
- For each reachable zone, there is a convex region of space visible to the camera

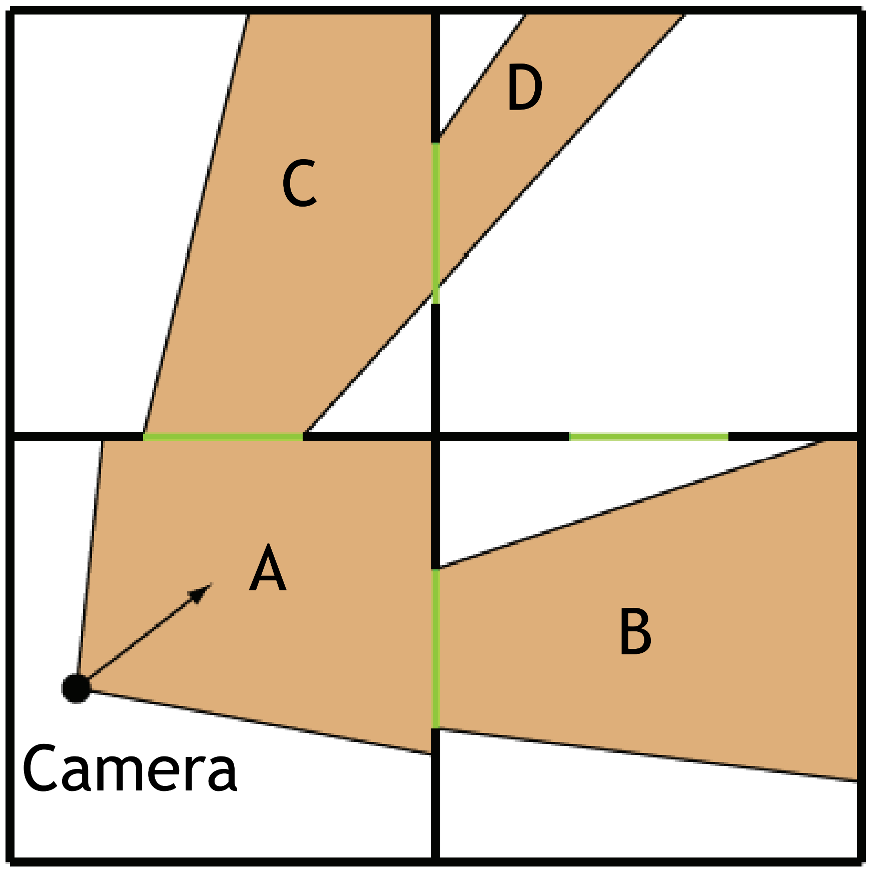
Portal Systems

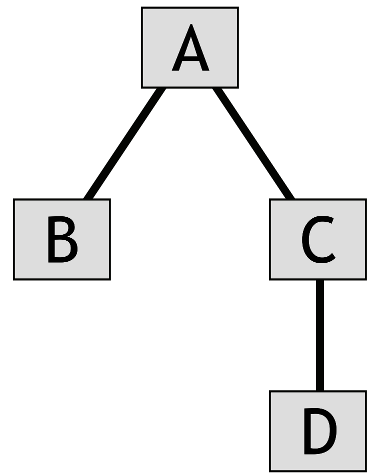


Portal Systems

- The visible regions form a tree structure
- The region in the zone containing the camera is the root of the tree
- Zones seen through *n* portals have regions at the *n*-th level in the tree

Portal Systems

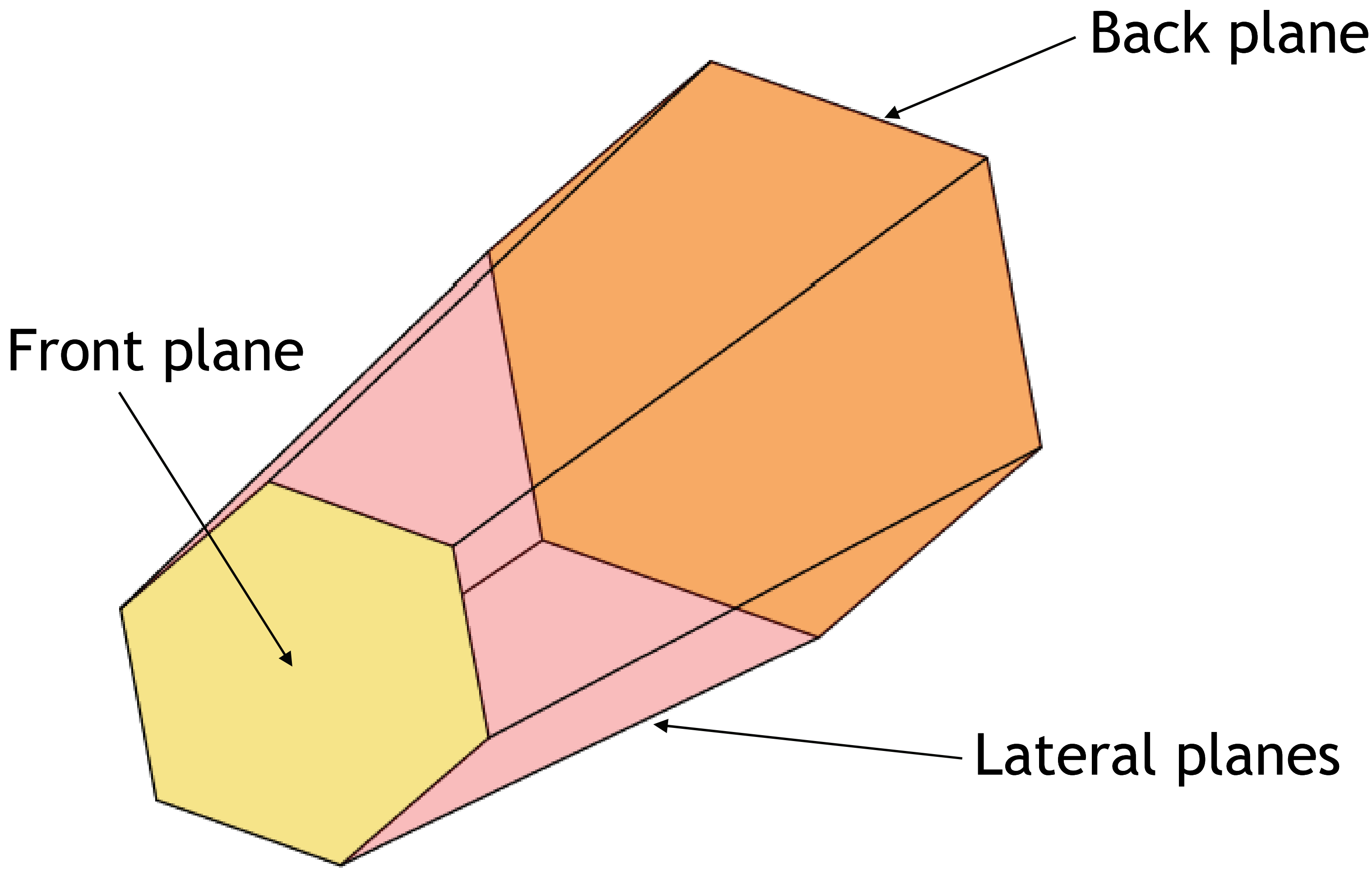




Regions

- We define a ***region*** to be a convex volume of space bounded by:
  - At most one front plane
  - At most one back plane
  - Any number of lateral planes
- Plane normals point inward

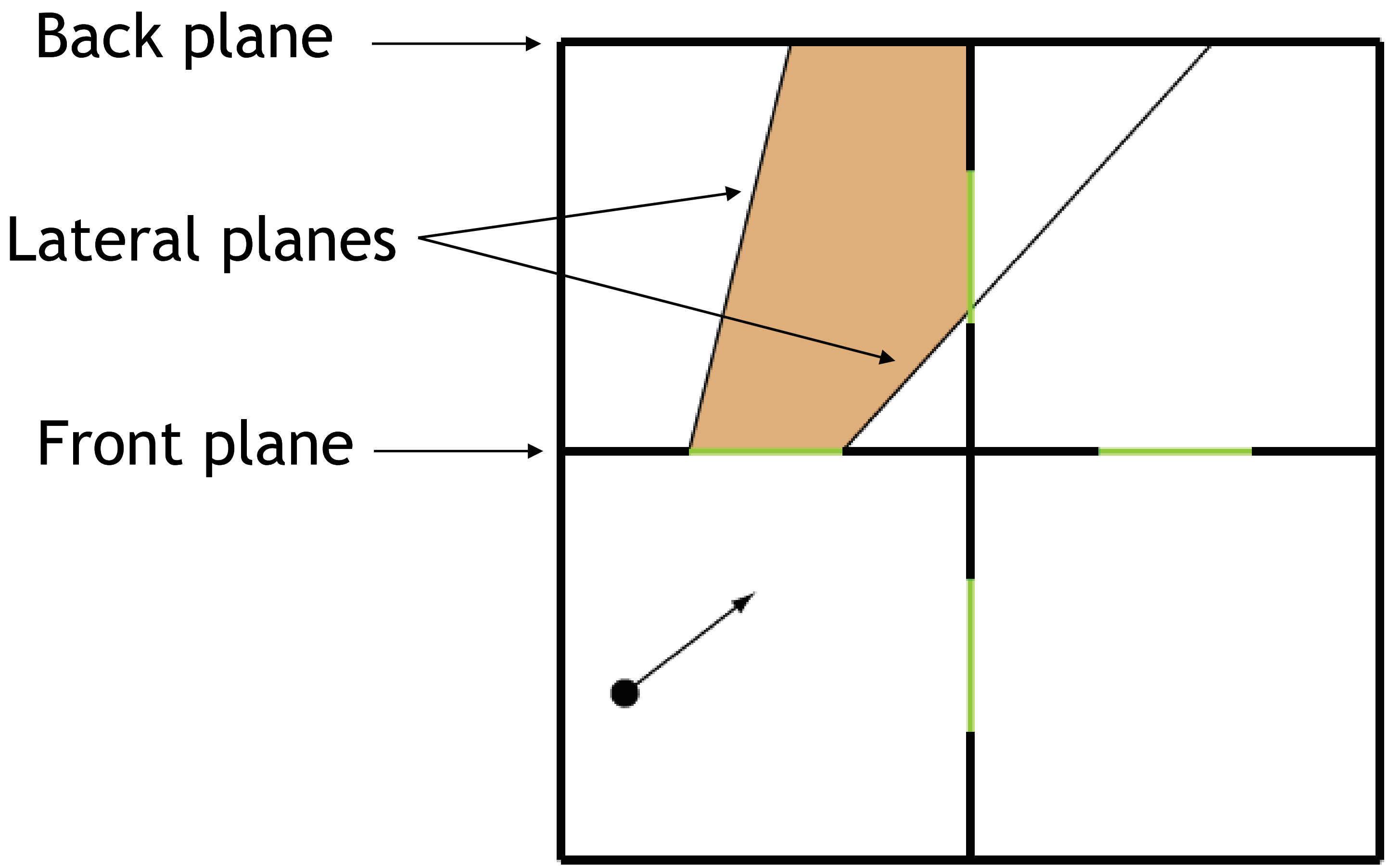
Regions



Regions

- Entrance portal determines the front plane
- Back plane determined by zone boundary
- Lateral planes determined by extrusion of clipped portal

Regions



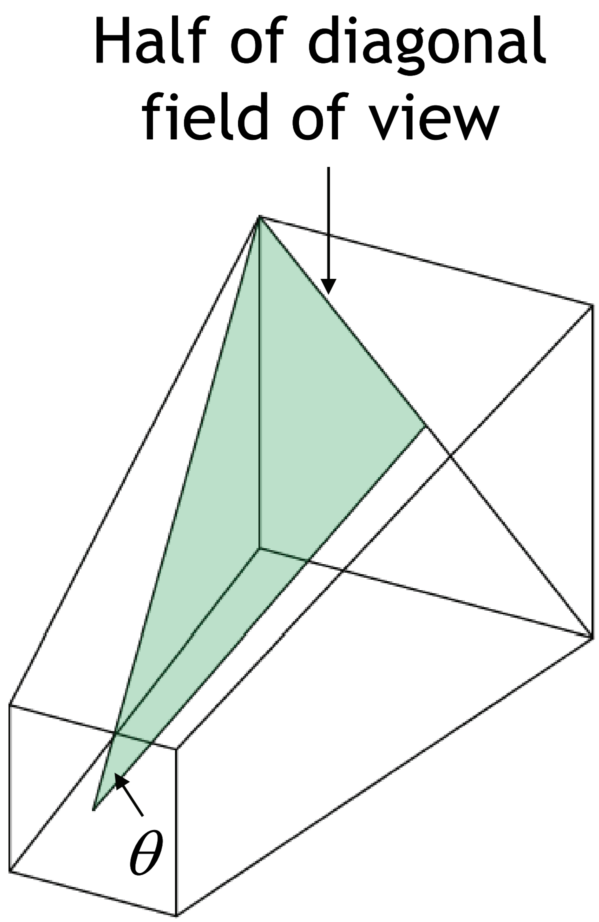
Building the Region Tree

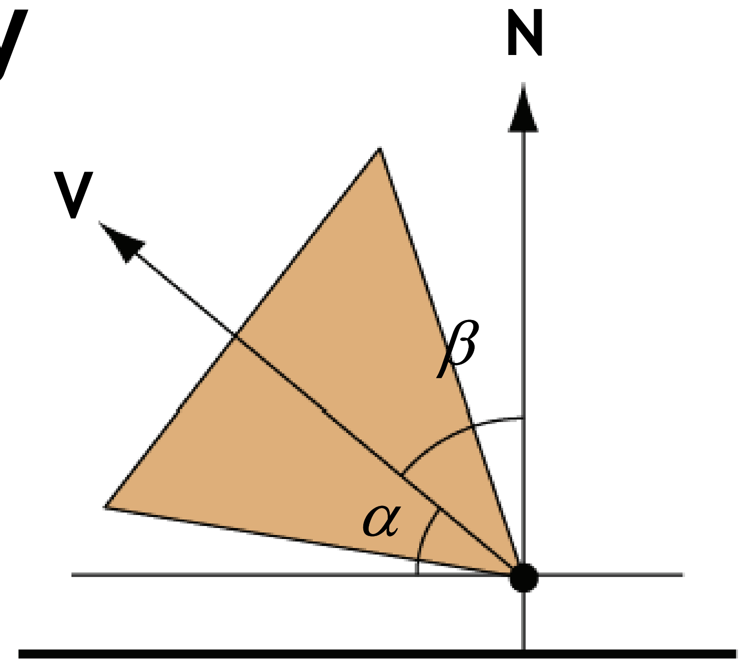
- Start with the zone containing the camera
- Then, recursively do…
  - Check portals leading out of current zone for visibility
  - Clip any visible portals to the bounding planes of the current region

Portal Visibility

- First calculate dot product *d* between camera view direction V and portal plane normal N
- Define θ to be half of the *diagonal* field of view
- If $d \geq \sin\theta$, then portal can't be visible

Portal Visibility





d = **V** · **N** = cos β = sin α

Portal only visible if
sin α < sin θ

Portal Visibility

- After field-of-view test...
  - Test portal bounding volume
  - If bounding volume visible, then clip portal polygon to region planes
  - *n*-sided portal clipped against *m* planes can have *n*+*m* vertices

Visible Object Set

- After region tree has been built...
  - Traverse the tree
  - Collect objects in each zone that intersect the visible regions corresponding to the zone
    - Use any frustum/bounding volume test, but test against region's planes
  - This is the visible object set

Region Classification

- Three types of region
  - "Camera region" refers to a region of space visible to the camera
  - "Light region" refers to a region of space reachable from a light source
  - "Shadow region" refers to a region of space from which shadows may extend into a camera region

Light Region Trees

- Portals can be used to construct illumination trees
  - Similar to the visibility tree constructed for the camera
  - One tree for each light source
    - Only recalculated when light moves
  - Each node in the tree corresponds to a convex region of space

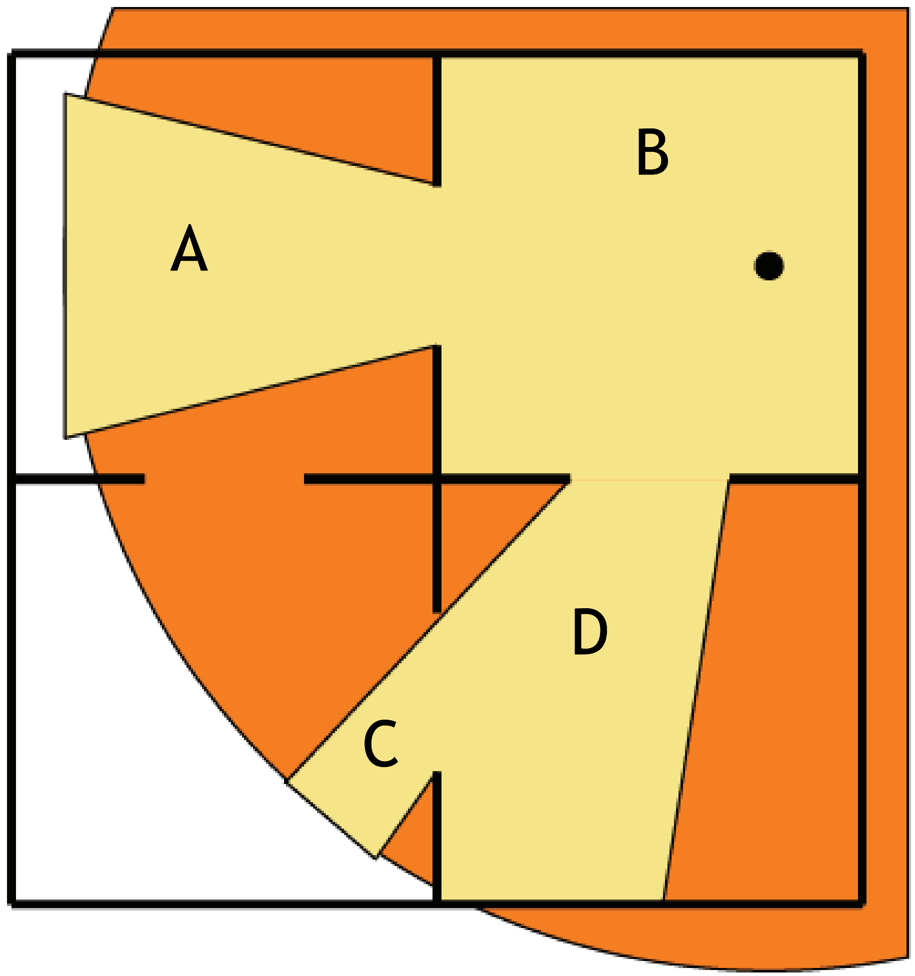
Light Region Trees

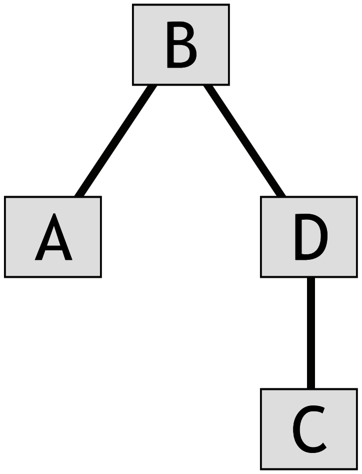
- Three fundamental light types
  - Point light
  - Spot light, special case of point light
  - Infinite (directional) light

Light Region Trees

- Point light
  - Omnidirectional
  - Has maximum range
  - Root illumination region bounded only be zone boundary and light's bounding sphere

Point Light Tree



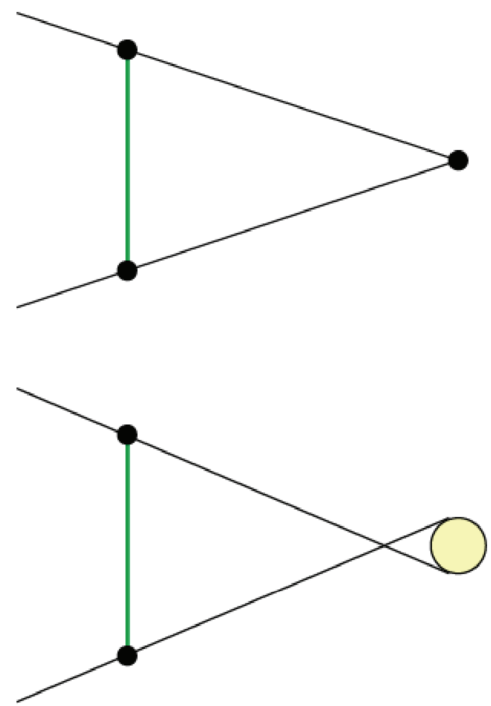


Spot Light Tree

- Spot light almost same as point light
  - Difference is the root node of the illumination tree
  - Spot light starts with a frustum, just like a camera does
  - Point light affects entire root zone

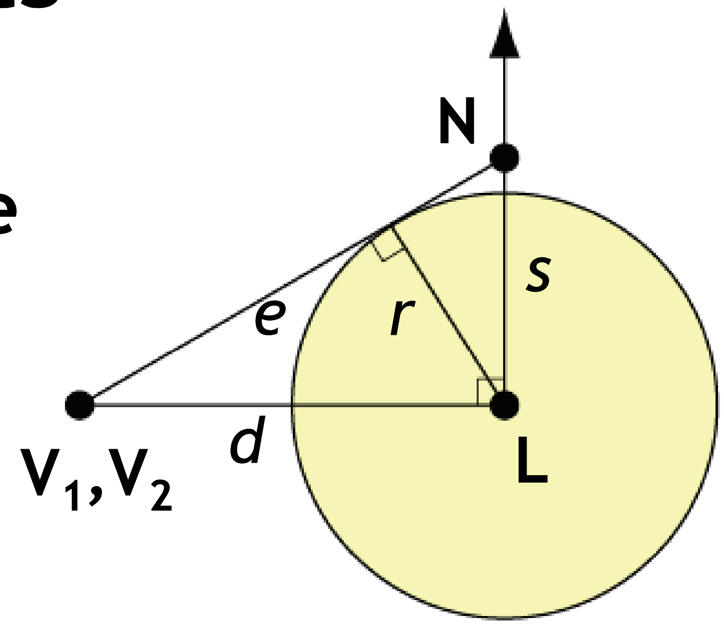
Area/Wiggle Lights

– Lateral planes need to be adjusted for area lights
– Same adjustment can be used to optimize 'wiggle' lights that can move within a small volume by removing need to recalculate regions



Area/Wiggle Lights

- Normally, a lateral plane is calculated using the portal edge V_1V_2 and the light position L
- Adjust for sphere of radius *r* by using the point L + *s*N



$$s = \frac{rd}{e}$$

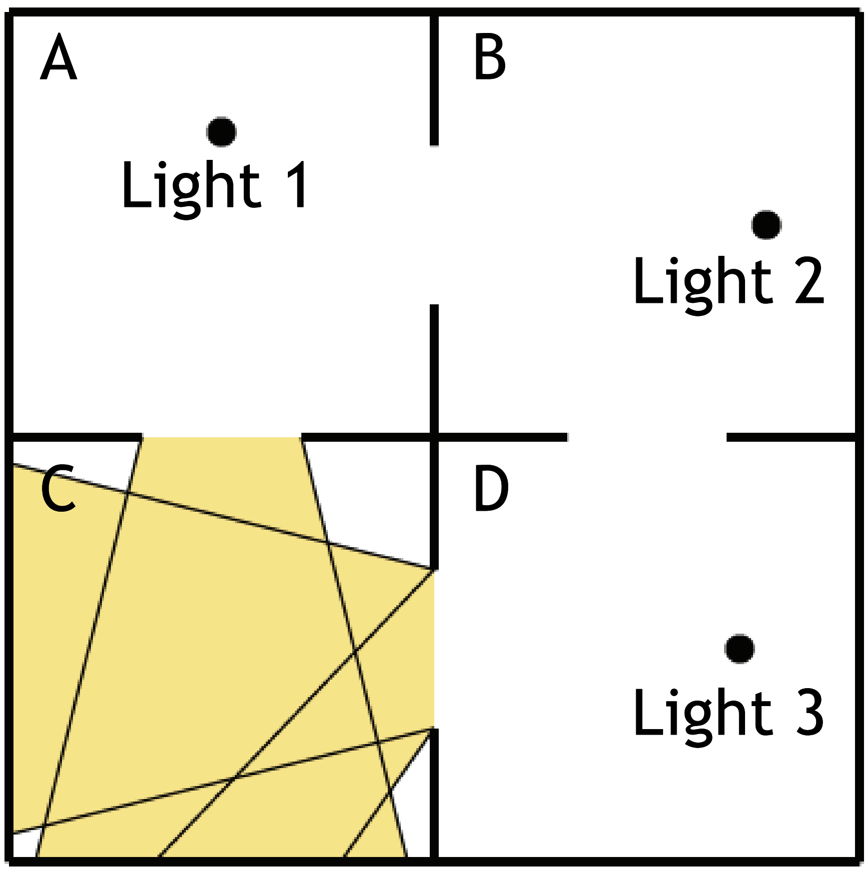
Infinite Light Tree

- Light rays parallel for infinite light
  - The lateral planes of each illumination region intersect at parallel lines
  - The extrusion of planes from a portal always goes in one direction instead of away from a point

Visible Light Determination

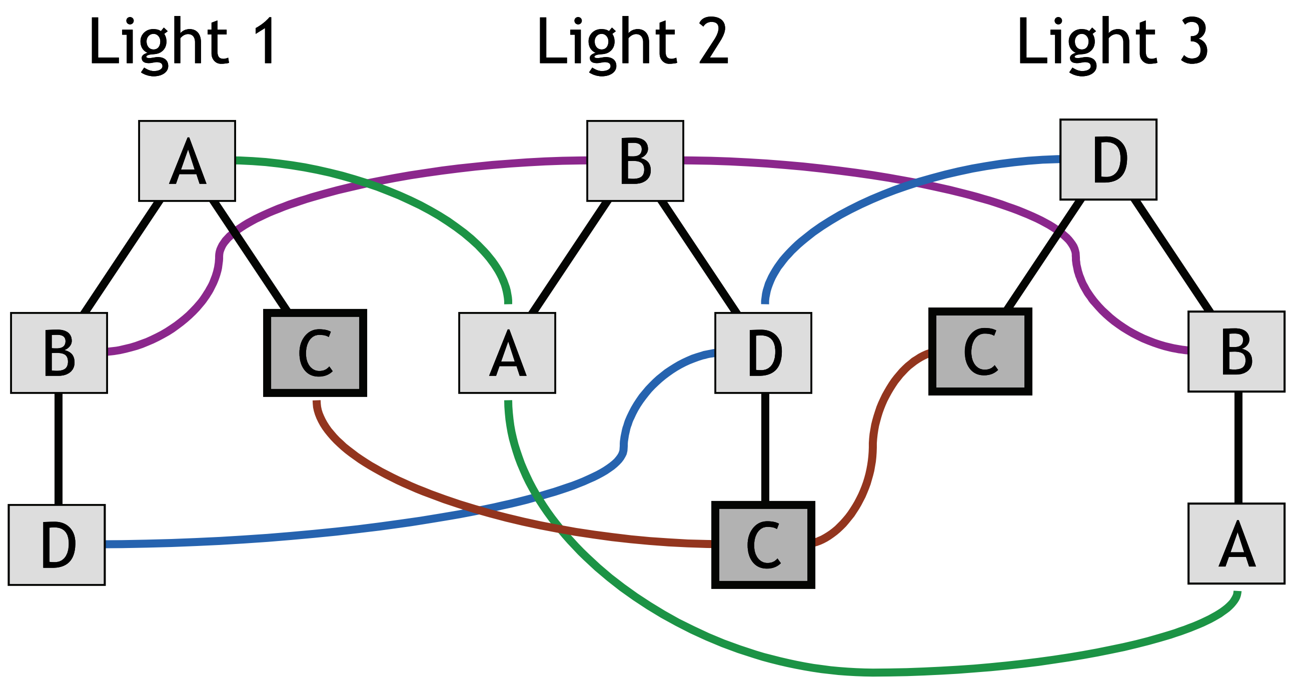
- Each zone keeps a linked list of light regions
  - One or more region nodes for each light that can shine into the zone
  - Each light region knows which light generated it

Visible Light Determination



For example, consider zone C

Visible Light Determination



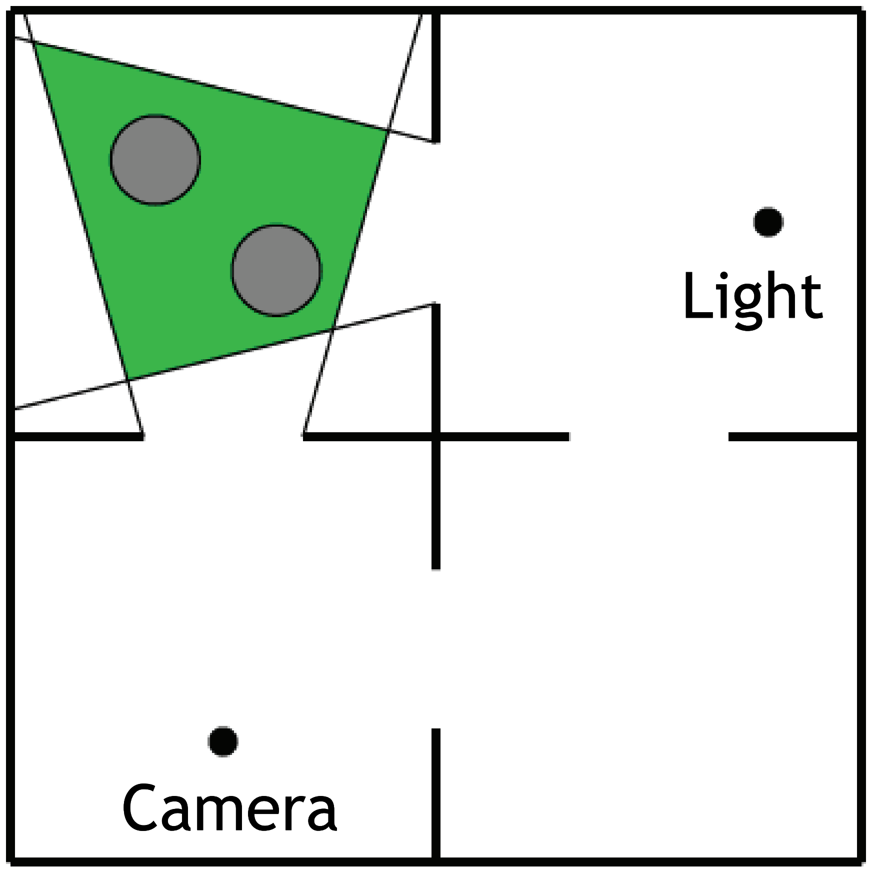
Visible Light Determination

- For any given zone, we can walk the linked list of light regions and collect unique lights
- Repeat process for all zones referenced in the camera's visibility tree
- We now have the set of visible lights

Illuminated Object Set

- Given one visible zone and one visible light shining into that zone…
  - Illuminated objects are those which intersect both a camera region and a light region

Illuminated Object Set



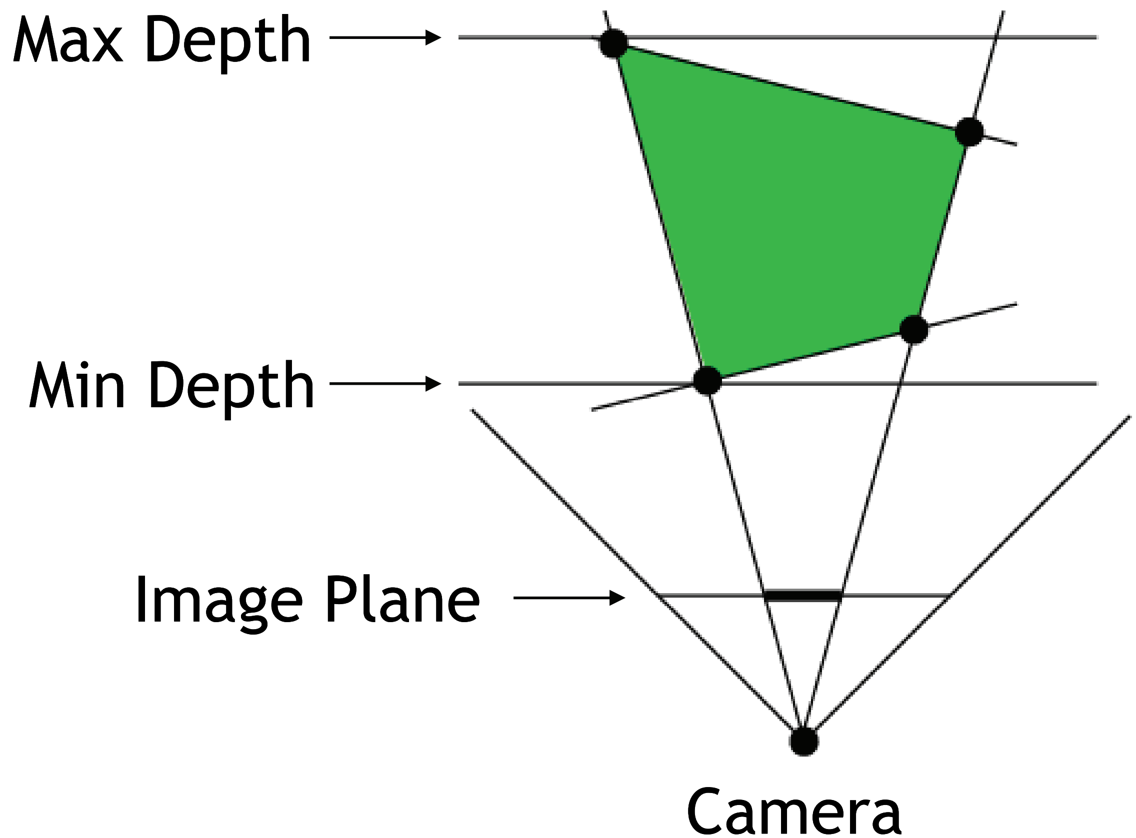
Illuminated Object Set

- Objects are often only partially within an illumination region
  - Lighting the whole object wastes rendering time due to extra fill
  - Fortunately, hardware provides an opportunity for optimization

Lighting Optimization

- Use hardware scissor rectangle
  - Calculate intersections of camera regions and light regions
  - Camera-space bounding box determines scissor rectangle
- GL_EXT_depth_bounds_test
  - Works like a *z* axis for scissor box, but a little different

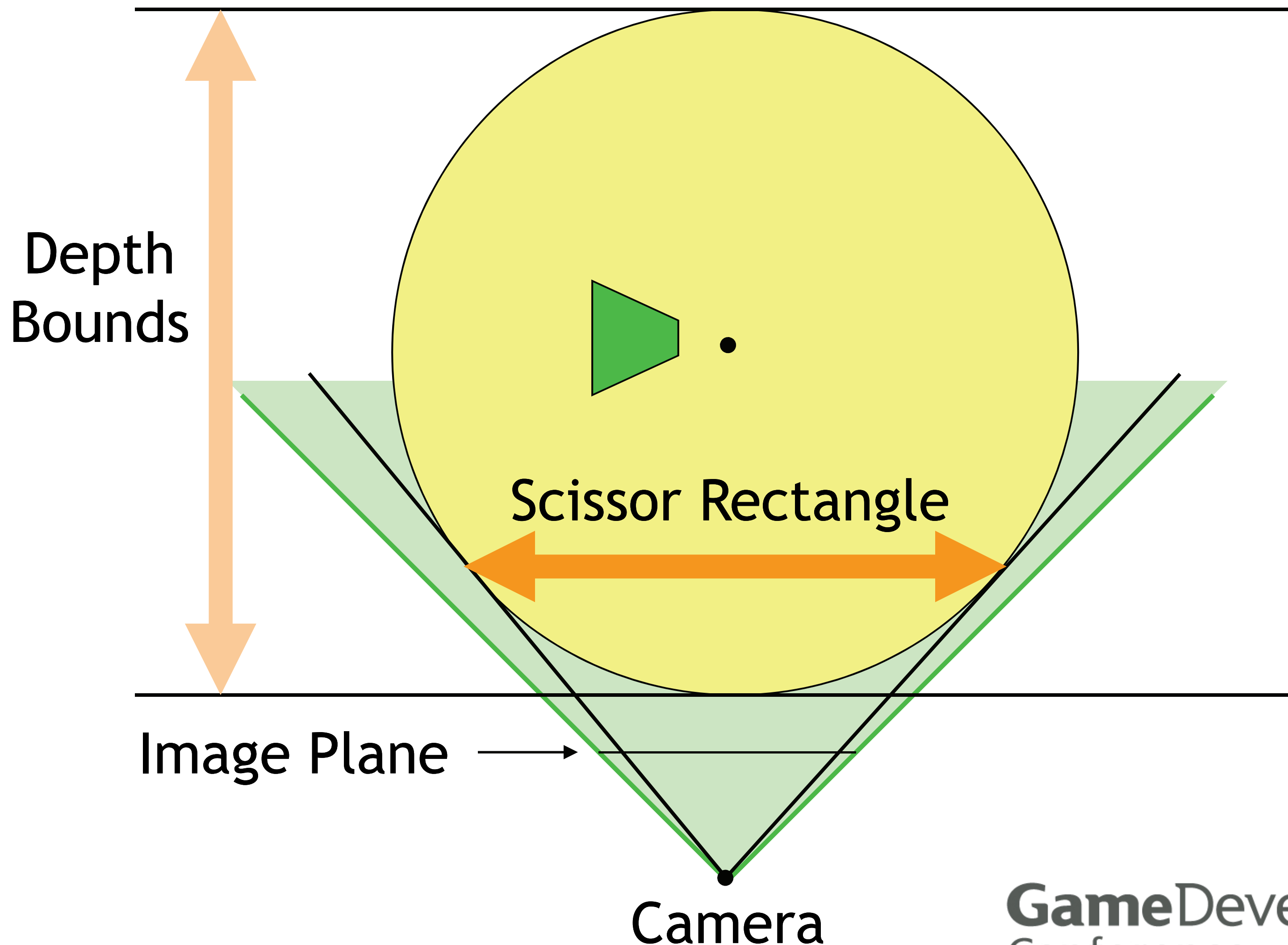
Lighting Optimization



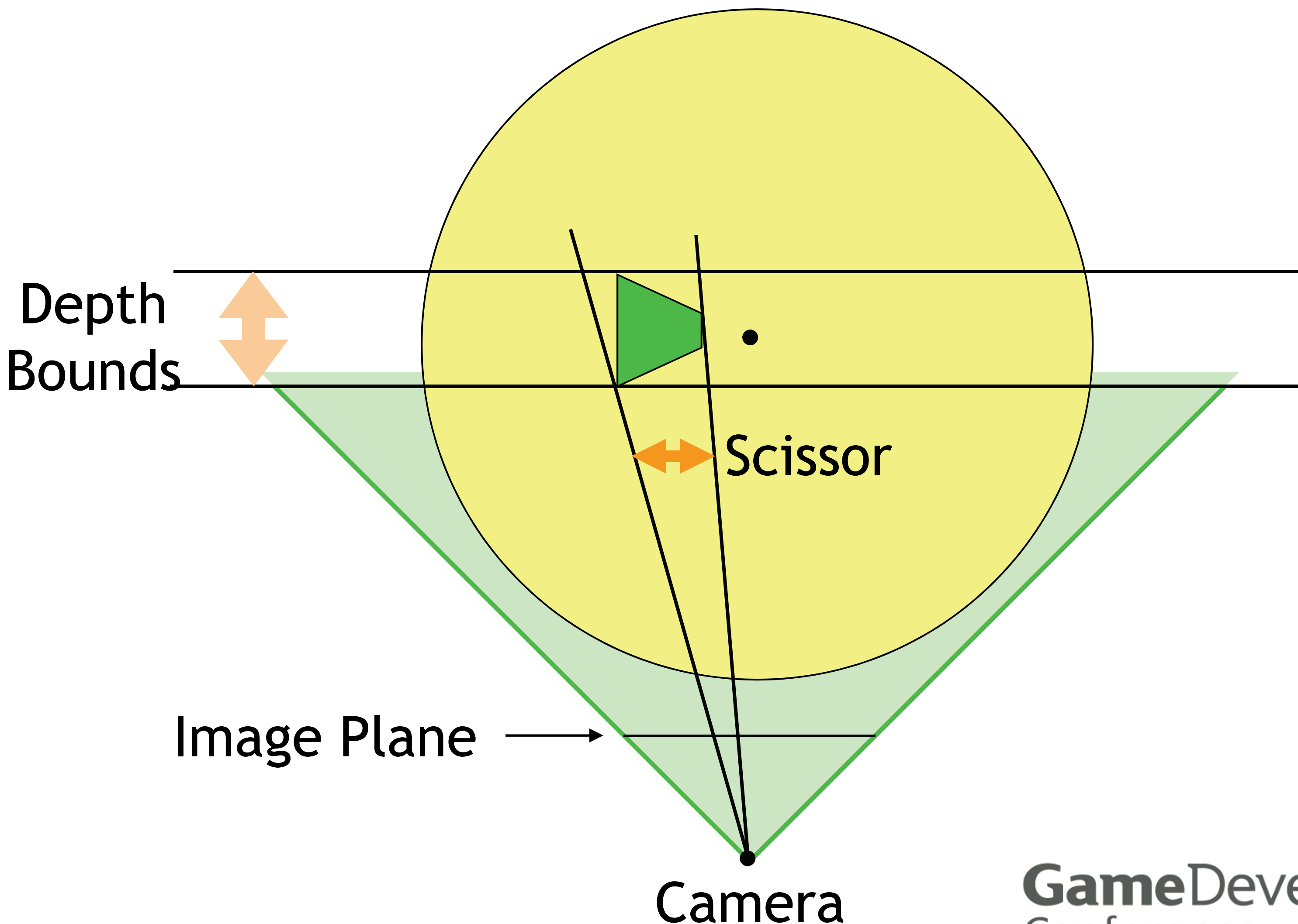
Lighting Optimization

- Scissor rectangle and depth bounds test
  - Limits rendering for a single light to the maximal visible extents
  - Can also be applied to stencil shadow volumes

Scissor and Depth Bounds



Scissor and Depth Bounds



Depth Bounds Test

- Let P be the projection matrix and let [$d_{\min}$, $d_{\max}$] be the depth range
- Viewport depth *d* corresponding to camera space *z* is given by

$$d = \frac{d_{\max} - d_{\min}}{2}\left(\frac{P_{33}z + P_{34}}{P_{43}z + P_{44}}\right) + \frac{d_{\max} + d_{\min}}{2}$$

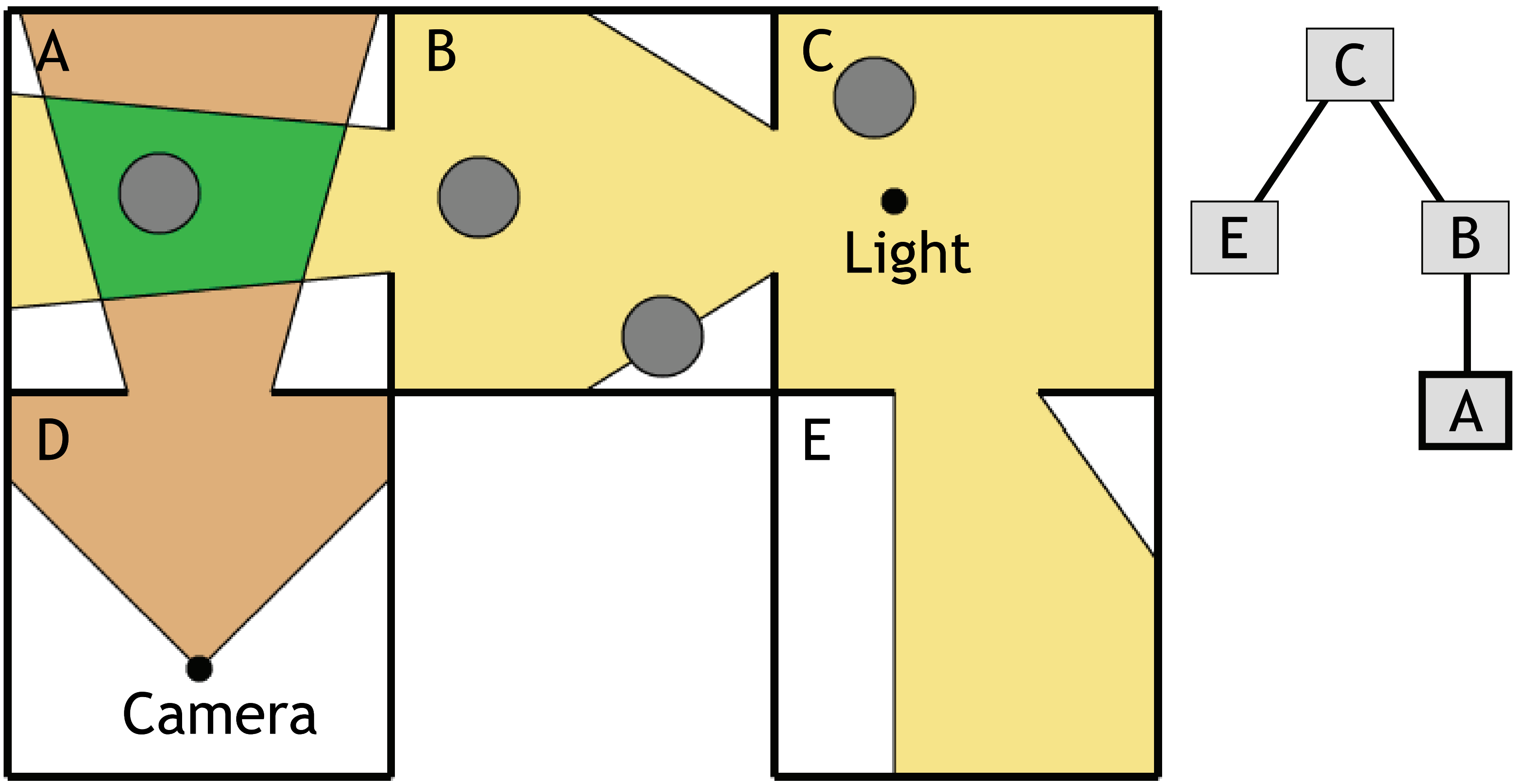
Shadow-Casting Object Set

- All objects in the illuminated set are also in the shadow-casting set
  - But an object doesn't have to be visible to be casting a shadow into one of the visible camera regions
  - The shadow-casting set is a superset of the illuminated set

Shadow-Casting Object Set

- Need to find objects between visible regions and light source
- We already have a structure in place to make this easy
- From a visible light region, walk up the light's illumination tree to the root

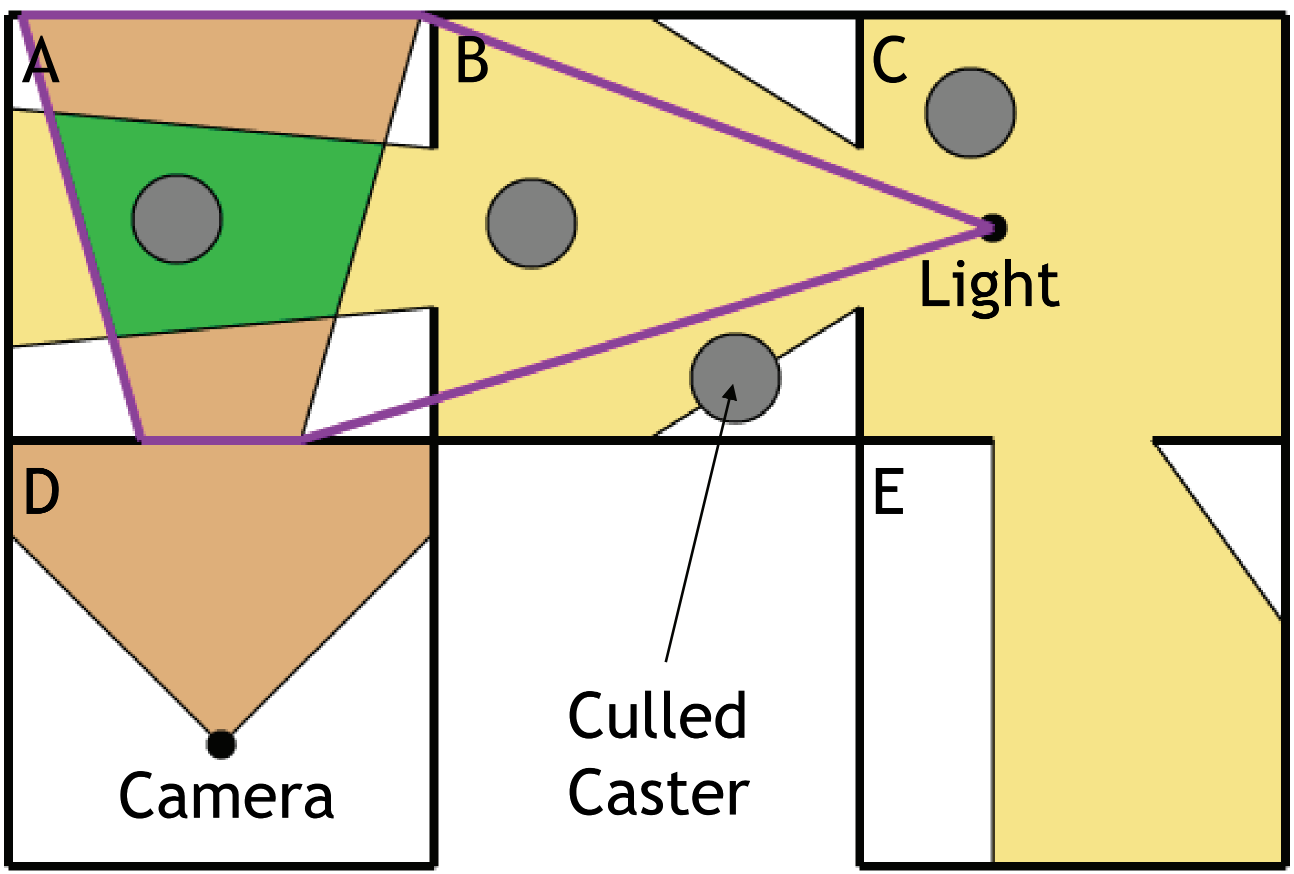
Shadow-Casting Object Set



Shadow Region

- Objects that can cast shadows into a visible camera region must:
  1) Lie in the camera region itself, or
  2) Lie in between the camera region and the light position
- The shadow region is the convex hull containing the camera region and the light position

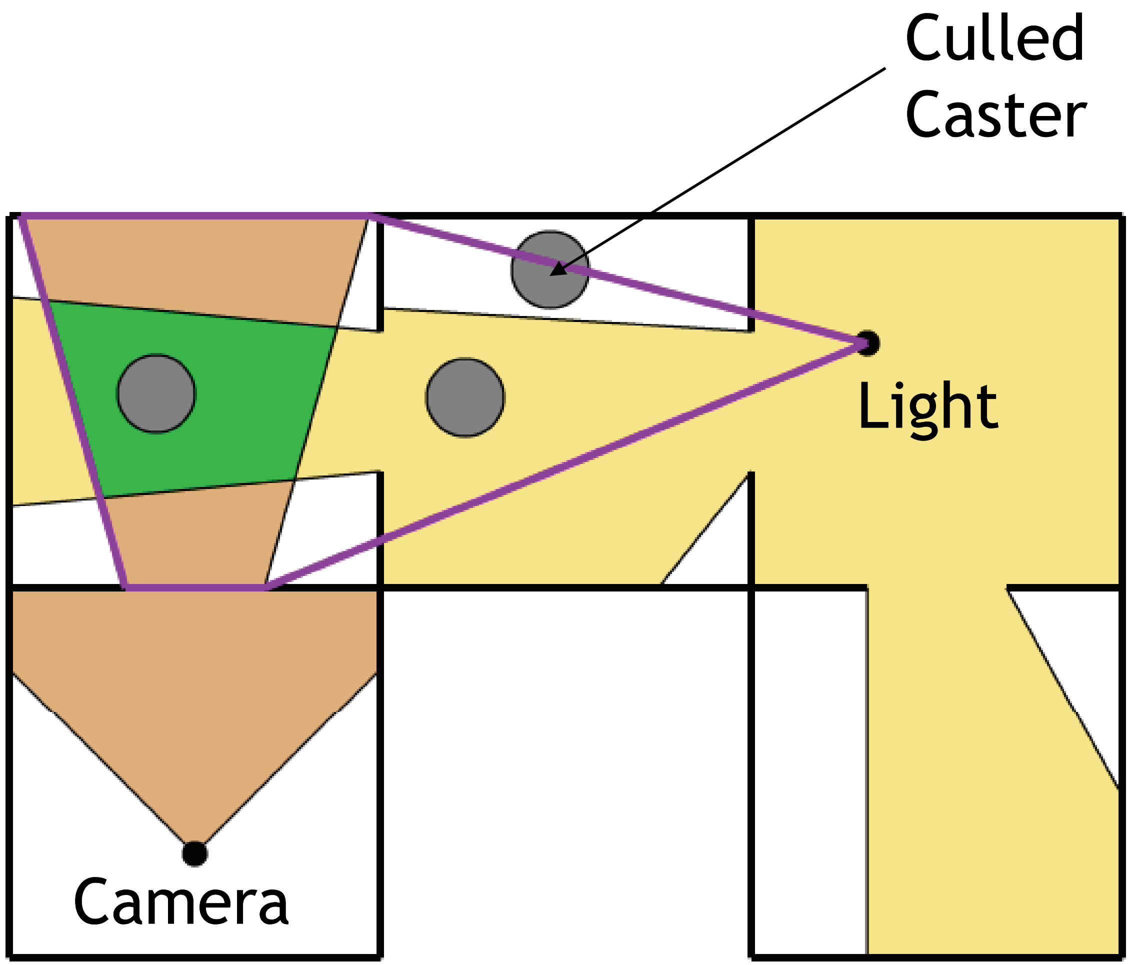
Shadow Region



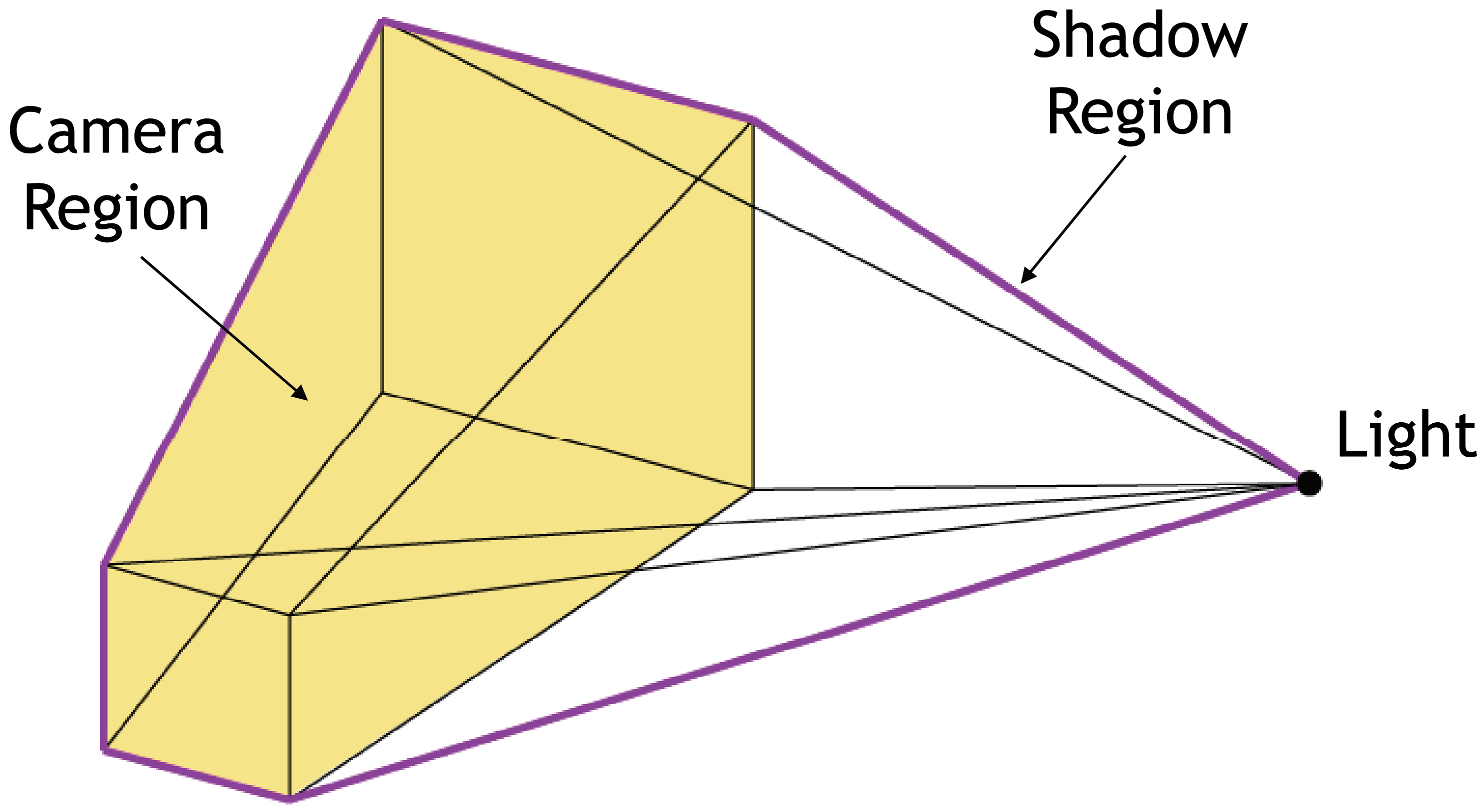
Shadow-Casting Object Set

- Collect objects in branch of illumination tree connecting visible camera region and light source
- But reject objects that don't intersect the shadow region AND their corresponding light region

Shadow Region



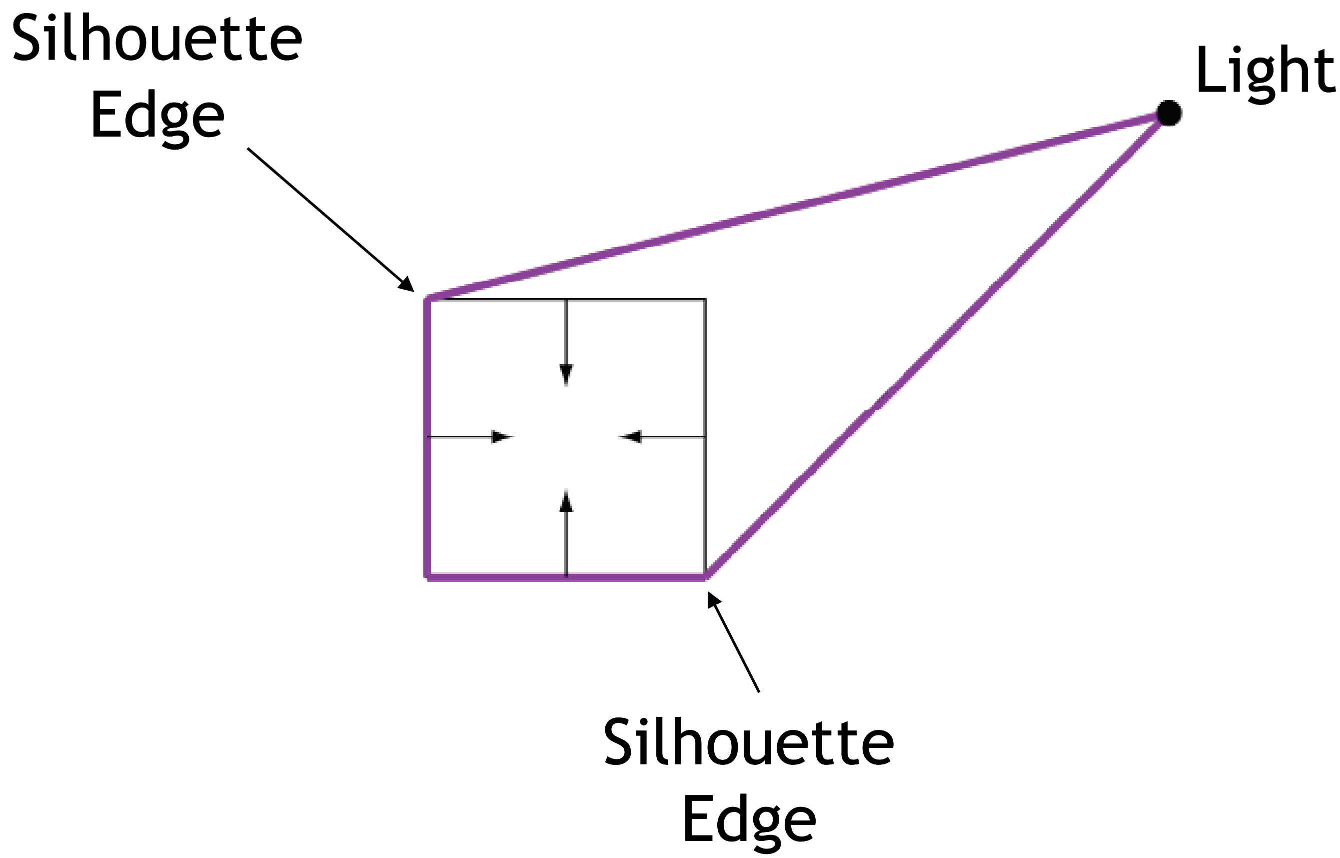
Shadow Region



Shadow Region

- Calculate dot product of each bounding plane of the camera region and the light position
- If positive, then the plane also bounds the shadow region
- Other shadow region bounding planes determined by camera region's silhouette

Shadow Region



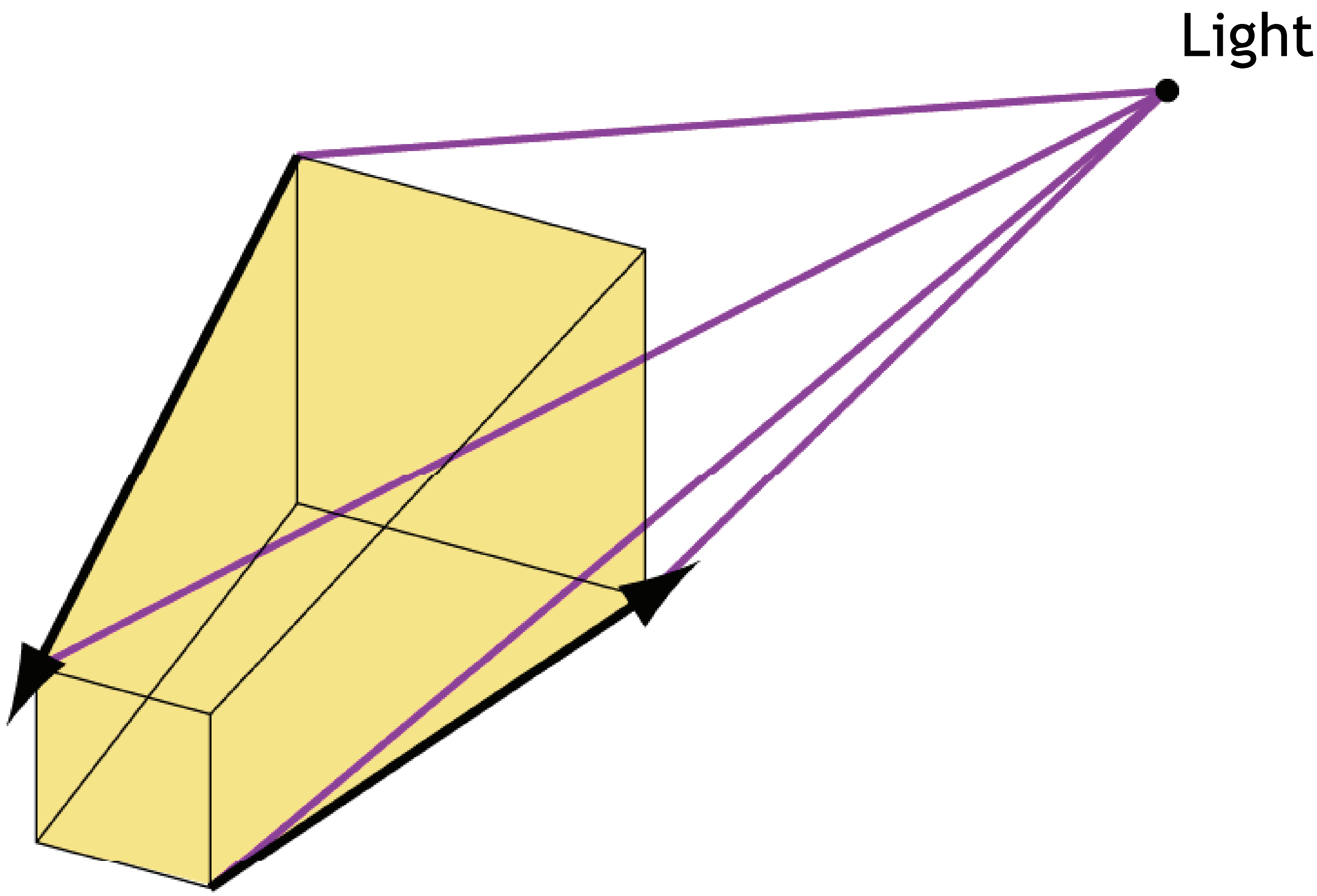
Shadow Region

- Lateral planes of camera region are wound CCW
- If two consecutive planes P_i and P_{i+1} have opposite-sign dot products with the light position L, then the edge between them is part of the silhouette

Shadow Region

- If $P_i \cdot L > 0$ and $P_{i+1} \cdot L \le 0$, then edge E should point away from camera
- If $P_i \cdot L \le 0$ and $P_{i+1} \cdot L > 0$, then edge E should point toward camera
- Bounding plane normal given by $(L - V) \times E$, where V is either edge endpoint

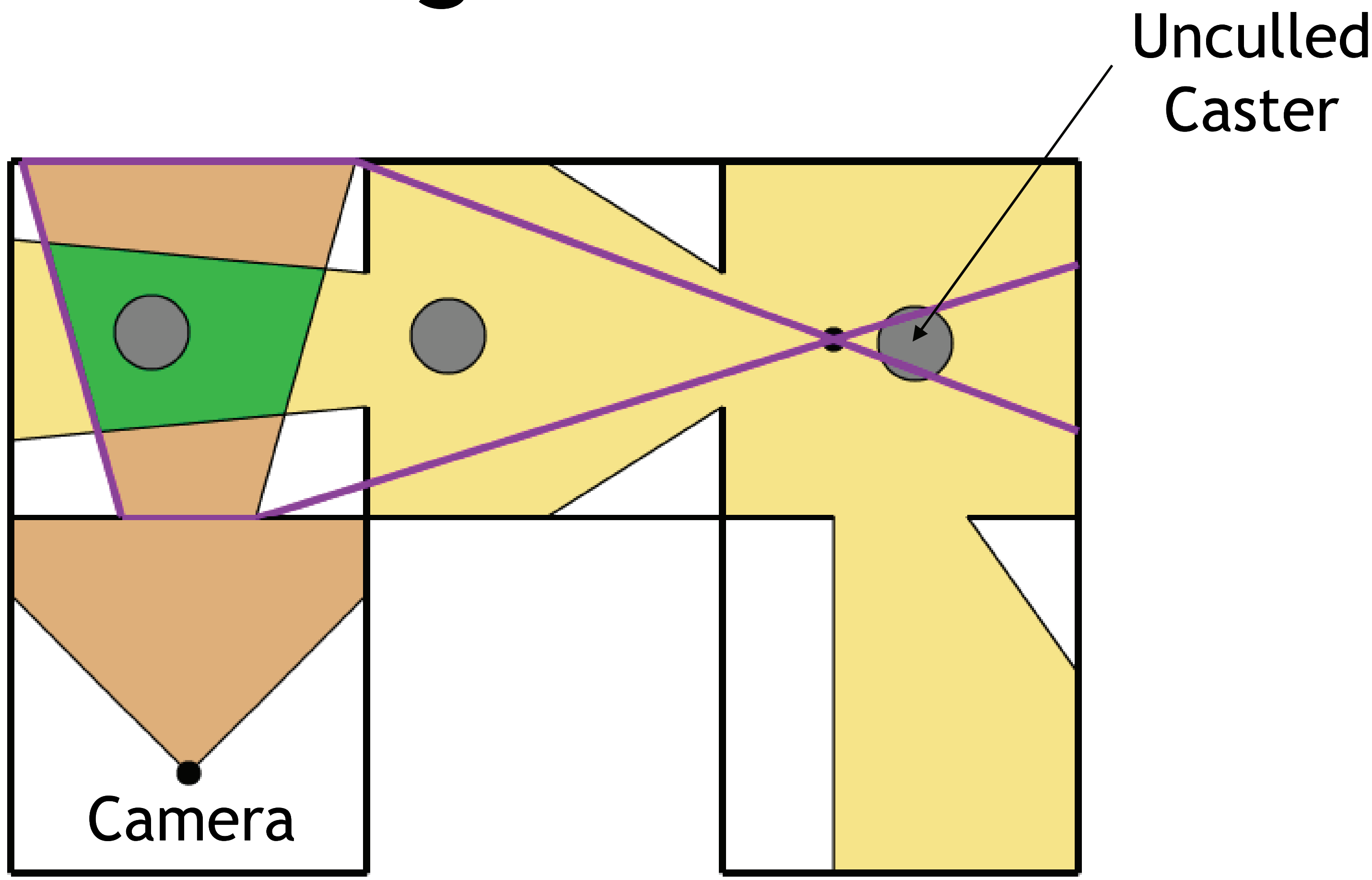
Shadow Region



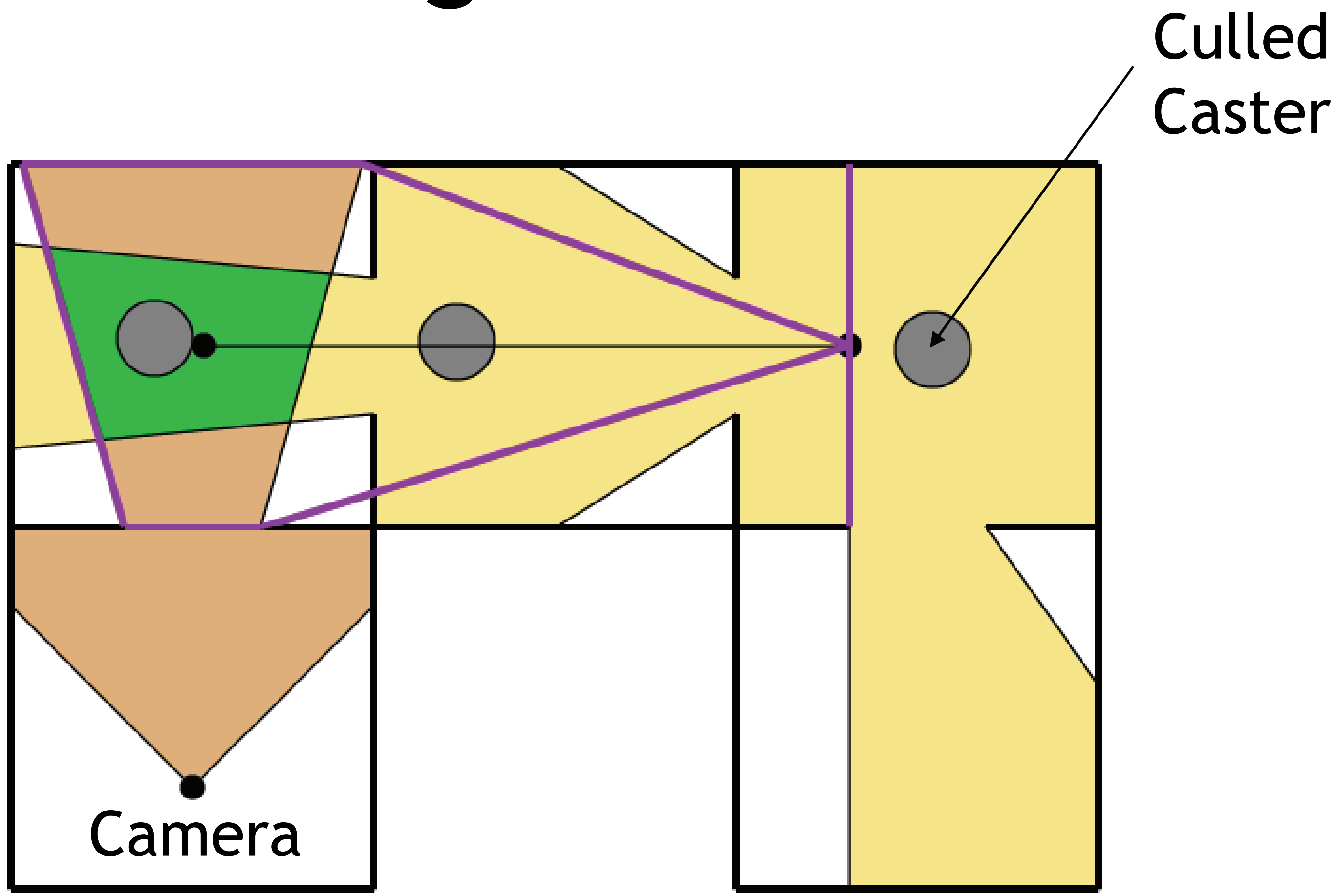
Shadow Region

- Also need to check edges between lateral planes and front/back planes
- Remember, vertices of front and back planes are wound CCW
- Adding a dummy front plane can help in cases of sharp point

Shadow Region



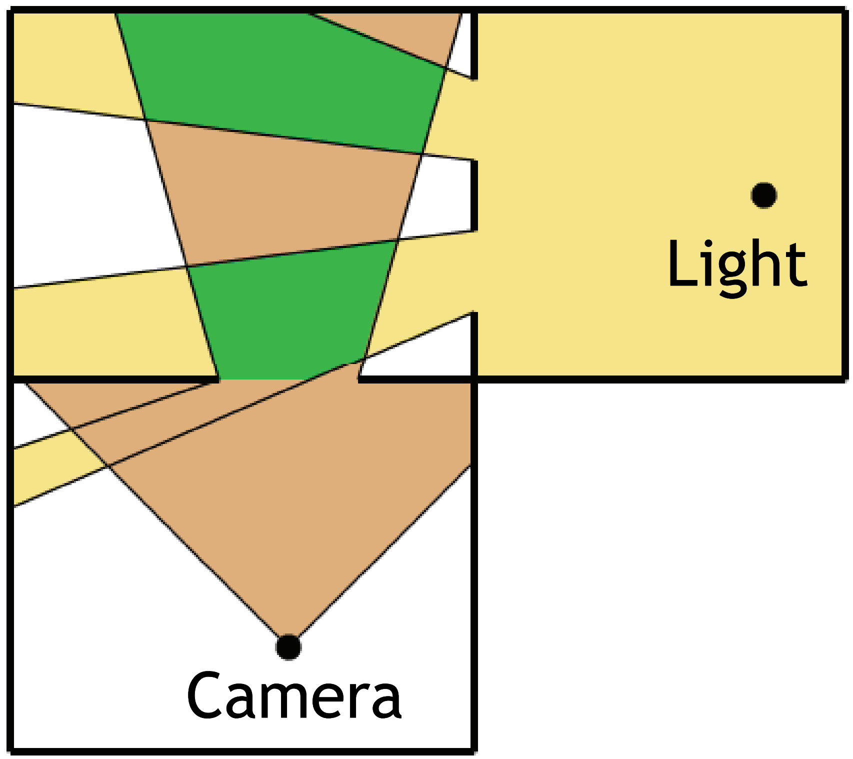
Shadow Region



Shadow-Casting Object Set

- What if multiple light regions intersect the camera region?
- What if one light region intersects multiple camera regions?

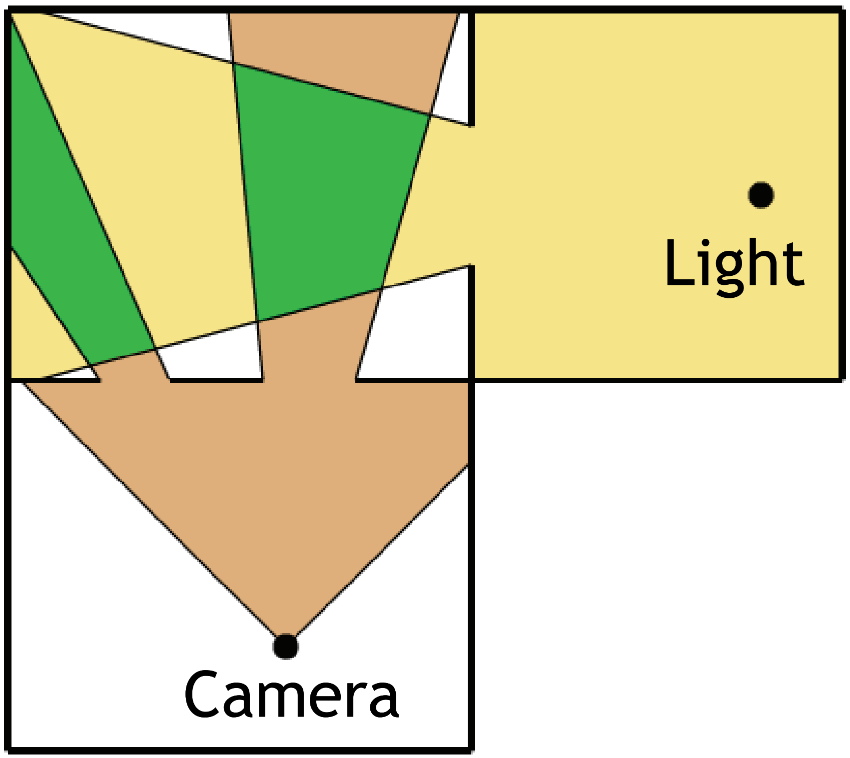
Multiple Light Regions for One Camera Region



Multiple Light Regions for One Camera Region

- The shadow region only depends on the camera region that each light region intersects
  - So the shadow region is the same for any pairing of light source and camera region
  - No need to take special action

Multiple Camera Regions for One Light Region



Multiple Camera Regions for One Light Region

- A separate shadow region needs to be constructed for each camera region
- There will be some overlap, so collect objects into some kind of container before rendering

Demonstrations

Questions?

- lengyel@terathon.com
- Slides available at

  http://www.terathon.com/