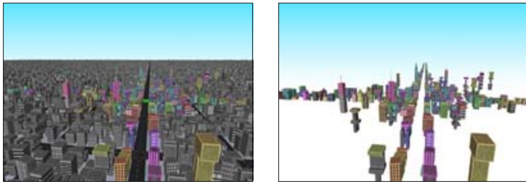
From-Region Visibility and Ray Space Factorization

Daniel Cohen-Or
Tel-Aviv University

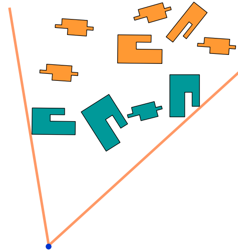


Overview

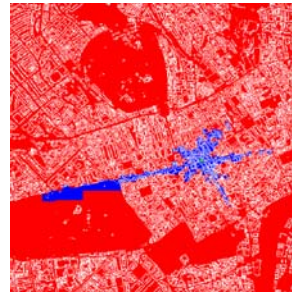
- Short introduction to the problem
- Dual Space & Parameter/Ray Space
- Ray space factorization (SIGGRAPH'03)

From Point Visibility

- Input:
  - Large scene
  - Viewpoint
- Output:
  - Set of visible objects from the viewpoint
- From the blue point only the blue objects are visible



From Point Visibility



From-region Visibility

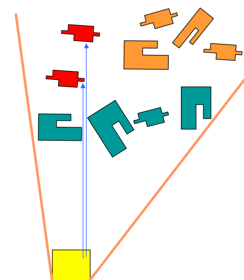
From Region Visibility

Compute the set of objects which partially visible from **anywhere** in the viewcell

A much harder problem:

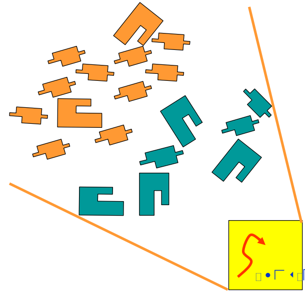
The red objects are now visible by the blue rays

4D problem for 3D scenes



Usages for From-Region visibility

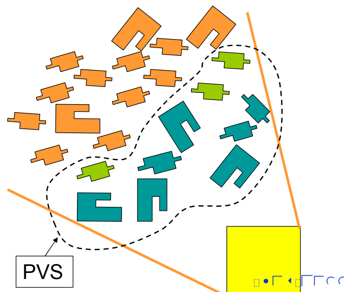
- Amortization:
  - Visibility data valid for many frames
  - Utilizes coherency between frames
- Web Systems:
  - Stream only the visible parts of the model
  - Overcomes latency



Conservative Visibility Sets

- Exact Visibility Set (VS):
  - Hard to compute
  - Superset for each individual viewpoint in the cell
- PVS (Potentially Visible Set): Conservative
  - Contains all the visible objects and maybe some occluded objects
  - Easier to compute

Conservative Visibility Sets



Conservative Visibility Sets (Cont.)

Computing a conservative PVS is the key point in designing an efficient visibility algorithm.

Tight PVS

Fast computation

Image-space Hierarchical Computation (Cont.)

- A top-down front-to-back traversal.
- Allows culling large parts of the scene
- Simple projection and simple image-space visibility test
- Conservative

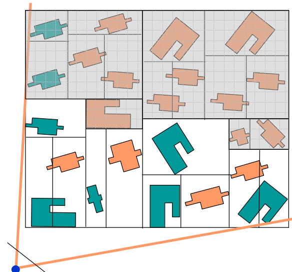
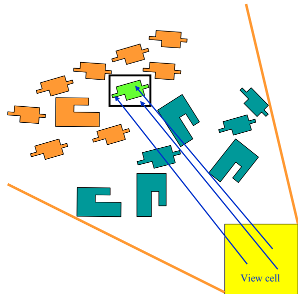


Image-space Hierarchical Computation (Cont.)

- Visibility test of cells is applied in every frame.
- Imposes overhead on rendering
- Using the same framework for from-region visibility will amortize the cost over many frames!
- But: we don't have a "center of projection" anymore – testing a cell-to cell visibility is harder.

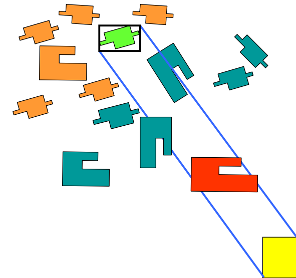
Straightforward solution: Sampling

- Problem:
  - Is the green object visible from the viewcell?
- Solution:
  - Sample some points from within the viewcell
  - Test if the green object is visible from each sampled point
- Con:
  - Not conservative
  - Slow!

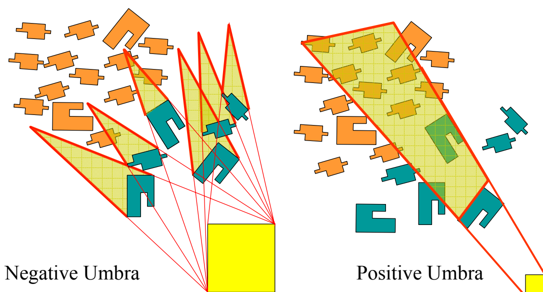


Strong Occlusion Test

- Algorithm:
  - Find objects occluded by a single convex occluder
  - Mark only such objects as hidden
- Pro:
  - Conservative solution
  - Fast
- Con:
  - Objects may be occluded only by a combination of other objects (Weakly Occluded)
  - Large viewcells



Umbrae of occluders

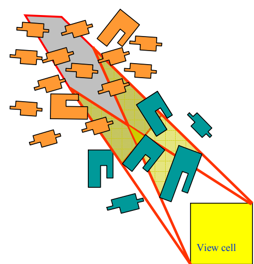


Occluder Fusion

- Visibility preprocessing with occluder fusion for urban walkthroughs
  - P. Wonka et.al. EGRW' 2000
- Virtual Occluder …
  - V. Koltun at.el. EGRW' 2000
- Conservative Volumetric Visibility with Occluder Fusion
  - G. Schaufler et.al. SIGGRAPH'2000
- Visibility Preprocessing using Extended Projections
  - F. Durand et.al. SIGGRAPH'2000

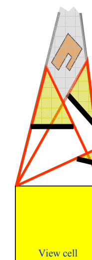
Occluder Fusion – Key Idea

- Merge umbrae that intersect
- Fuse small umbrae into larger aggregated umbrae



Intersecting Umbrae - problem

Doesn't capture all the cases of occluder fusion

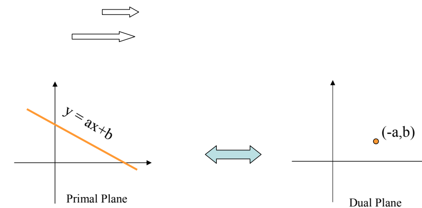


Ray (parameter) space techniques

- All rays hitting an object define a footprint in parameter space.
- Boolean set operations on footprints determine visibility.
- Exact - captures all the cases of occluder fusion.

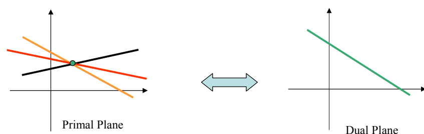
Simple Dual Space

- Simple duality transformation in 2D:
  - L : y = ax+b L^*: (-a,b)
  - p: (a,b) p^* : y = -ax+b



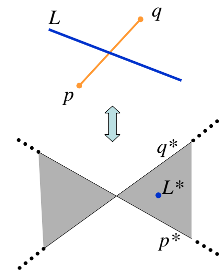
Simple Dual Space

- All lines passing through a point are mapped to a line



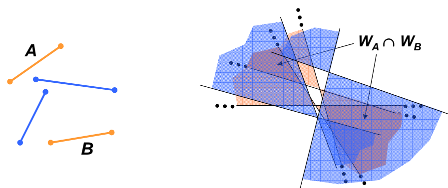
Basic Dual Space (Cont.)

- The lines that intersect a line-segment produce a *double-wedge* in dual-space
- We can encode <u>all</u> the lines that intersect a segment into some footprint in dual-space



Parameter Space Footprint

- Is segment A mutually visible from segment B ?
  - $W_A \cap W_B$ describes all possible sight-lines
  - The union of footprints of the occluder segments is the aggregated occlusion of these segments



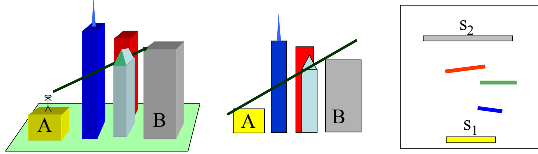
Parameter space - discussion

- Advantages:
  - Easy to implement with Boolean set operations
  - Great for occluder fusion – provides exact solution
- Problems:
  - The footprint is unbounded – can't be discretized and implemented in hardware
  - No simple extension to 3D rays

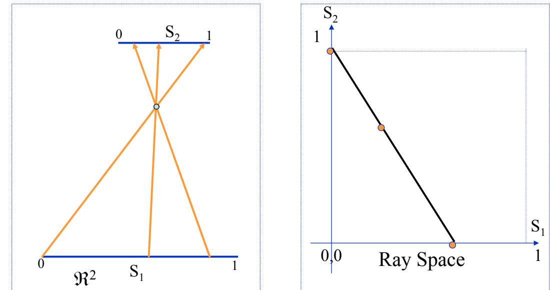
Hardware Accelerated

[Koltun, Cohen-Or and Chrysanthou, EGRW2001]

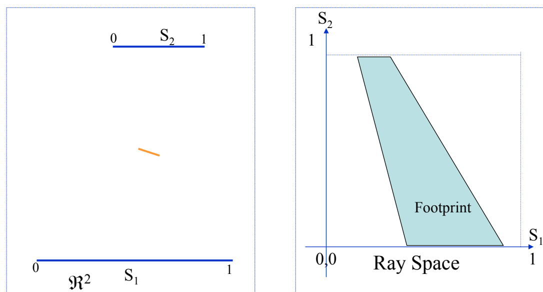
- Put a plane lying on the viewcell A and the target cell B
- For 2.5D occluders: A & B are mutually visible iff their upper rims are mutually visible
- Reduces the 2.5D problem into planar visibility test



Parameter Ray Space

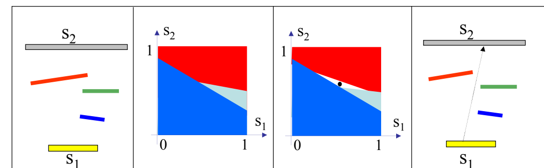


Parameter Ray Space



Hardware Accelerated Occlusion Test

- Render all the footprint polygons onto parameter space
- Check whether the frame buffer is fully covered
- Conservativeness - draw only fully covered pixels



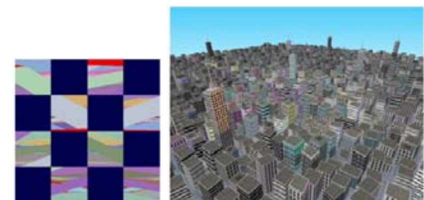
Discussion

- This parameterization is bounded
- Can be efficiently used with graphics hardware

- The parameterization is valid only within the shaft.
- Need to construct different parameter space for each occludee – which means each occluder is processed many times.
- 2.5D occluders only.
- Conclusion: we need something better…

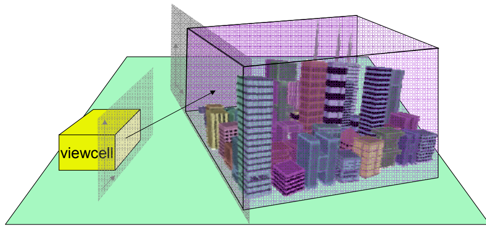
Ray Space Factorization (Siggraph'03)

- Use a bounded parameter space
- One global parameter space
- Each occluder is processed once
- Support 3D scenes
- Fast using graphics HW



The dimensionality of the from-region problem

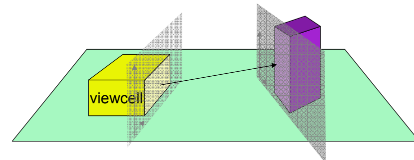
From-Region visibility is 4D



The dimensionality of the from-region problem

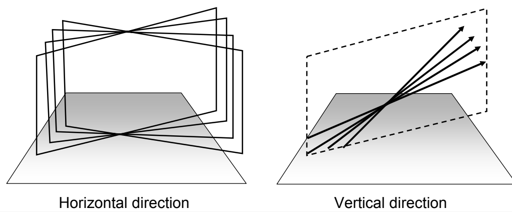
From-Region visibility is 4D

A ray exists the viewcell through a 2D surface and enters the target region through a 2D surface



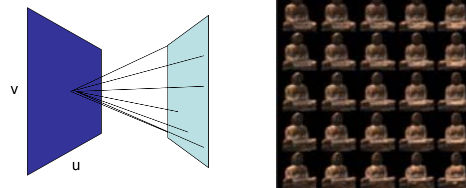
Our Factorization

We factor the 4D visibility problem into horizontal and vertical components



Lumigraph/light-field

A 2D grid of 2D images



Our Main Contribution

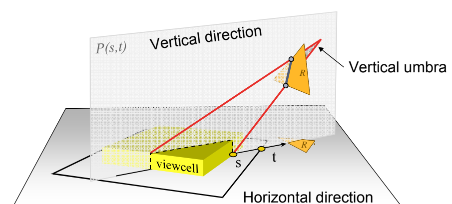
Our factorization:

- Exploits vertical coherence
- Maps to the graphics card

Algorithm Overview

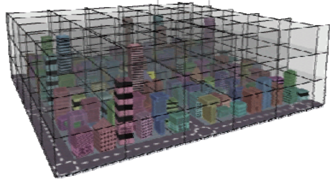
Per Object:

- Parameterization of vertical slices
- Umbra encoding



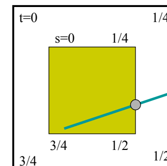
Algorithm Overview (Cont.)

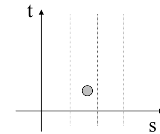
Top-down, front-to-back traversal of a KD-tree hierarchy
Objects in visible leafs serve as occluders



Parameterization in 2D

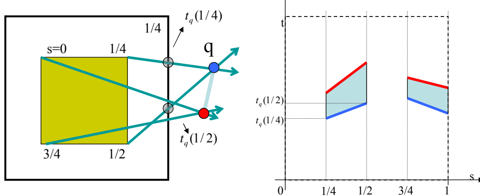
- Two concentric squares
- Parameters (*s,t)* are associated with the inner and outer squares (0≤s,t≤1)





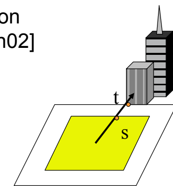
A Footprint of a Segment in 2D

A footprint of a 2D segment is several 2D polygons

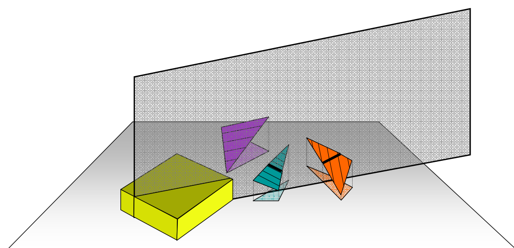


Extension to 3D

- 2D solution is not enough
  - Can't tell whether an object is really occluded
- Plucker Coordinates
  - The curse of dimensionality
  - Very slow, no hardware realization [Bittner and Prikryl 01, Nirenstein02]

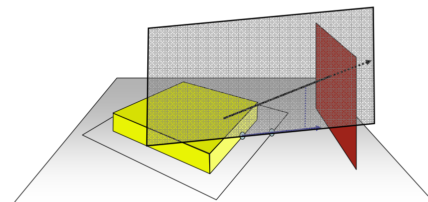


Within a Vertical Slice



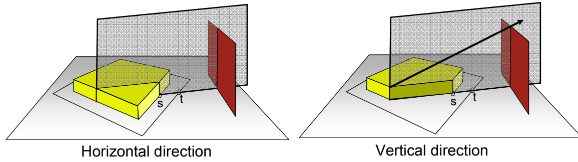
Within a Vertical Slice

A ray that leaves the viewcell has a 2D horizontal direction
Each horizontal direction (s,t) defines a *vertical-slice*
Within the slice the ray has a 2D vertical direction



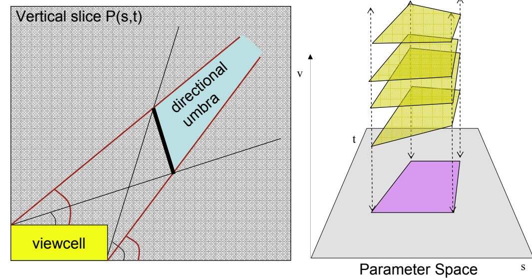
Horizontal and Vertical directions

We need to encode more than a single value per slice…



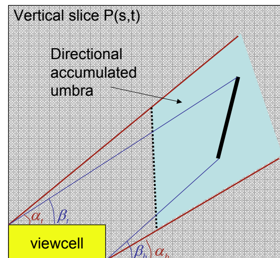
Umbra Encoding

Encode supporting and separating angles



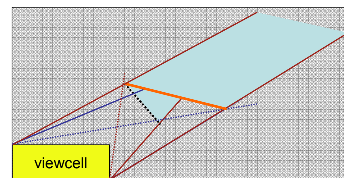
Testing Visibility

In parallel (in all slices) test occlusion by comparing supporting angles



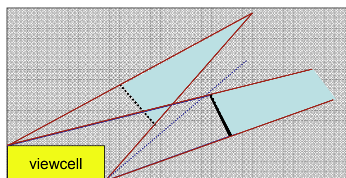
Umbrae Merging

Augment (fuse) the aggregated umbra if the directional umbra intersects



Umbrae Merging (Cont.)

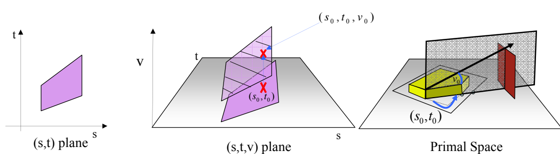
Otherwise, create another umbra entry.
If there are too many discard it.



A simple case

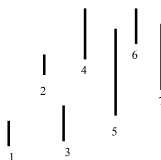
To make it clearer, let's look at a simpler case.

Here we just encode the top-elevation angle



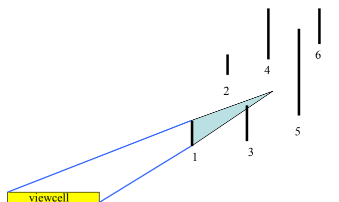
Merging Umbrae (within the slice)

- Process objects in front-to-back order
- Maintain the aggregated umbrae



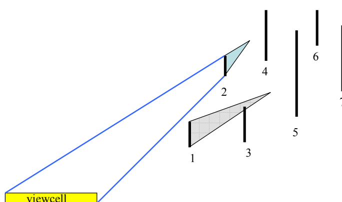
Merging Umbrae

- Process objects in front-to-back order
- Maintain the aggregated umbrae



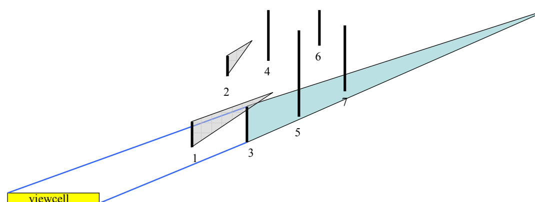
Merging Umbrae

- Process objects in front-to-back order
- Maintain the aggregated umbrae



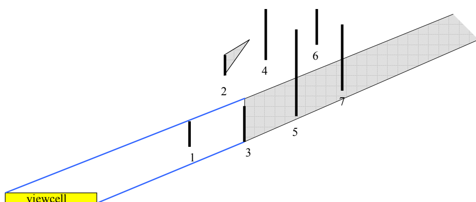
Merging Umbrae

- Process objects in front-to-back order
- Maintain the aggregated umbrae



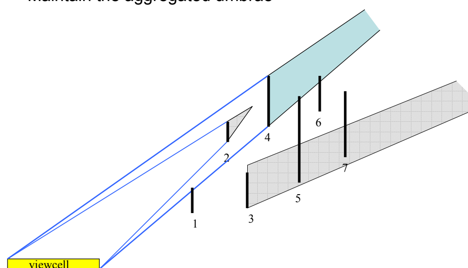
Merging Umbrae

- Process objects in front-to-back order
- Maintain the aggregated umbrae



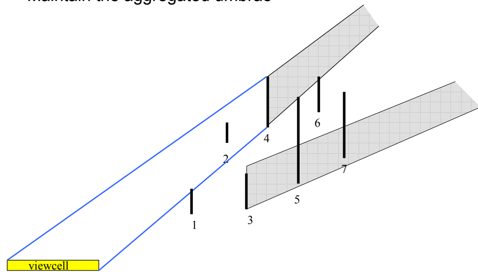
Merging Umbrae

- Process objects in front-to-back order
- Maintain the aggregated umbrae



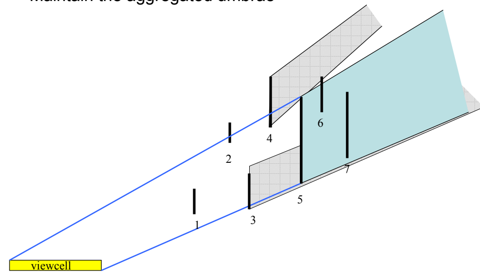
Merging Umbrae

- Process objects in front-to-back order
- Maintain the aggregated umbrae



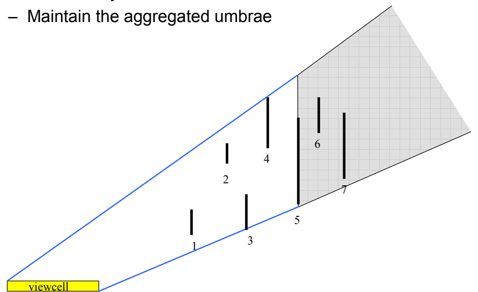
Merging Umbrae

- Process objects in front-to-back order
- Maintain the aggregated umbrae



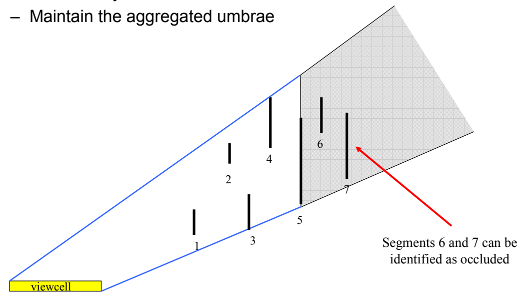
Merging Umbrae

- Process objects in front-to-back order
- Maintain the aggregated umbrae



Merging Umbrae

- Process objects in front-to-back order
- Maintain the aggregated umbrae

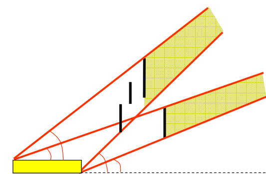


Pixel-shader

- Performs all the directional operations simultaneously over all the pixels of the occluder footprint.
- The pixel resolution defines the degree of conservativeness.

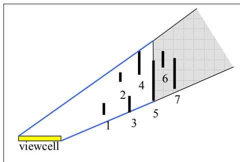
A Multi-layer Occlusion Map

- Each pixel is associated with a series of supporting and separating angles pairs used by the pixel-shader operation
- The number of umbrae is limited

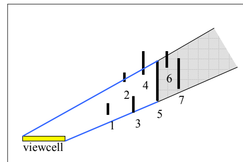


A Single Aggregated Umbra

- Maintain a single aggregated umbra
- The algorithm is still conservative



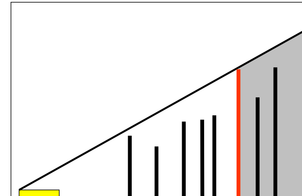
Final aggregated umbra using two aggregates



Final aggregated umbra using a single aggregate

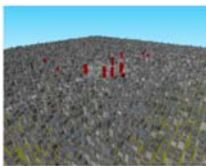
A Single Value – 2.5D scene

- Aggregated umbrae always intersect
- Fast on common graphics card

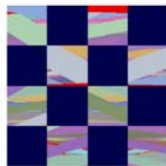


2.5D Example

- The red skyscrapers (left) have large top supporting angles
- Their red footprints are visible => the skyscrapers themselves are visible



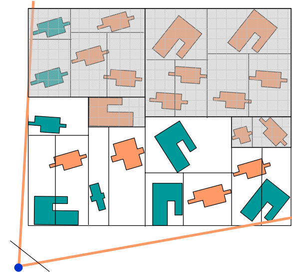
Primal Space



Parameter Space

Image-space Hierarchical Computation

- A top-down front-to-back traversal.
- Allows culling large parts of the scene
- Simple projection and simple image-space visibility test
- Conservative



Ray-space Hierarchical Computation

- A top-down front-to-back traversal.
- NOT-Simple projection and simple ray-space visibility test
- Conservative

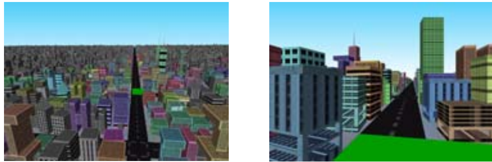
Results

Random Urban Model

Vienna 2000 Model

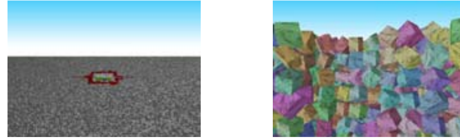
Box Field Model

Results – Random Urban Model



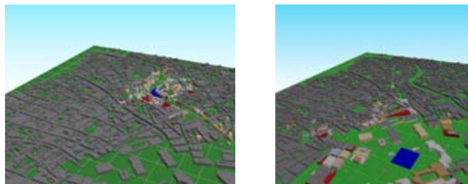
Cell size	Off-junction viewcells		In-junction viewcells	
	Time	PVS / VS	Time	PVS / VS
3	0.31	2412 / 1760	0.40	8832 / 6824
9	0.41	2840 / 2632	0.51	12184 / 9072
14	0.58	3592 / 2894	0.96	13576 / 9184
20	0.71	4568 / 2928	1.56	18304 / 9888
25	0.78	5224 / 2960	2.01	21608 / 10072

Results – Box Field Model



Cell size	Near viewcell		Far viewcell	
	Time	PVS / VS	Time	PVS / VS
5	0.93	4864 / 1312	3.22	35456 / 14224
10	1.10	6032 / 1424	3.45	37072 / 14256
15	1.27	7184 / 1552	3.59	38912 / 14304
20	1.39	8176 / 1632	3.71	40080 / 14416
30	1.87	13136 / 2400	4.04	43248 / 14672

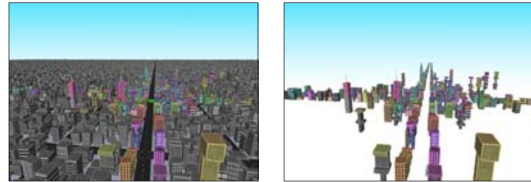
Results – Vienna 2000 Model



Small viewcells			Large viewcells		
Size	Time	PVS / VS	Size	Time	PVS / VS
3	0.18	964 / 424	50	0.36	3396 / 502
14	0.19	1216 / 514	100	0.45	4424 / 744
25	0.22	1546 / 592	150	0.58	5720 / 930

The End

Thank You



Preliminary results

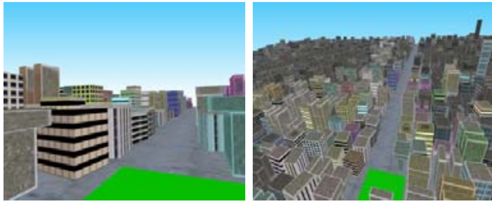
- The algorithm is extremely fast if implemented with hardware support.
- Even faster with new hardware (pixel shader operations, occlusion flag)

No. of trapezoids	Total Times (ms)	Footprint times (ms)	Frame Buffer	
			Total time	Read
4.7K	479	101	374	351
18.8K	433	87	341	320
80.1K	605	123	476	447
325.9K	834	164	663	622
1,308.7K	895	177	710	668

Performance Table

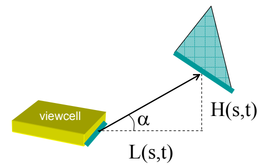
Figure 8: A scene composed of T & H shaped buildings.

The End



(a) (b)
City with T & H shaped buildings along a street. (a) is taken from within the view-cell (in green) and (b) from a birds eye (gray buildings are hidden)

Non-linearity of parameterization



$$\tan\alpha = \frac{H(s,t)}{L(s,t)}$$

We approximate this rational function by a linear function

Handling arbitrary triangles

Horizontal component: parameterize each visible edge

Inside directional slice: as before!