Impostors for Interactive Parallel Computer Graphics

Orion Sky Lawlor

olawlor@uiuc.edu

2004/4/12

Overview

- Impostors Basics
- Impostors Research
- Parallel Graphics Basics
- Parallel Impostors
- Parallel Planned Work
- Graphics Planned Work

Thesis Statement

- **Parallel impostors can improve performance and quality for interactive computer graphics**
  - **Impostors** are 2D standins for 3D geometry
  - **Parallel impostors** are impostor images computed on a parallel server
  - **Interactive** means there’s a human watching and controlling the action with fast response times

Importance of Computer Graphics

- “The purpose of computing is insight, not numbers!” R. Hamming

- Vision is a key tool for analyzing and understanding the world
- Your eyes are your brain’s highest bandwidth input device
  - Vision: >300MB/s
    - 1600x1200 24-bit 60Hz
  - Sound: <1 MB/s
    - 96KHz 24-bit stereo
  - Touch: <100 per second
  - Smell/taste: <10 per second

Impostors

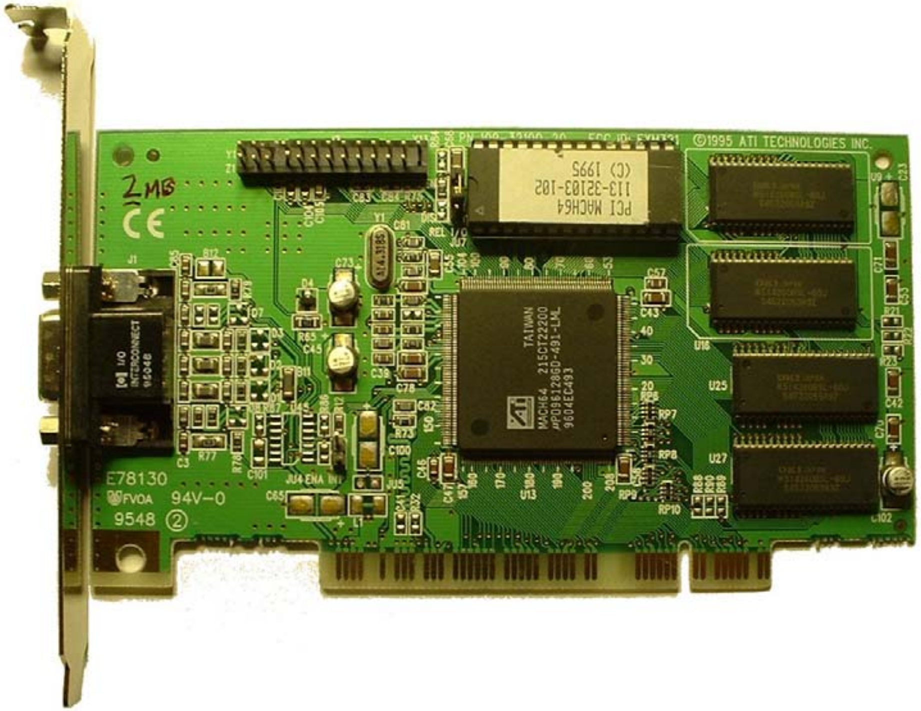
Fundamentals
Prior Work

Impostors

- Replace 3D geometry with a 2D image
- 2D image fools viewer into thinking 3D geometry is still there
- Prior work
  - Pompeii murals
  - *Trompe l'oeil* ("trick of the eye") painting style
  - Theater/movie backdrops
- Big limitation:
  - No parallax

[Harnett 1886]

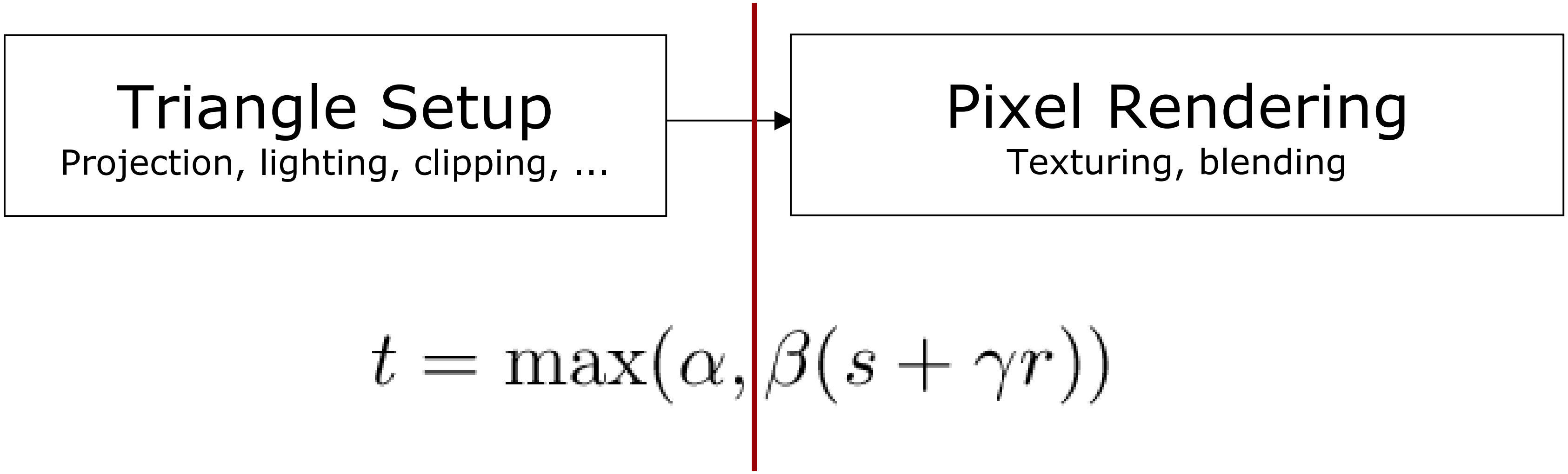
Graphics Cards



- Draws *only* polygons, lines, and points
- Supports image texture mapping, transparent blending
- Portable, usable OpenGL software interface

- Interactive graphics now *means* graphics hardware
  - SGI pioneered modern generation (early 1990’s)
  - Explosion of independent companies (1995)
  - Consumer hardware vertex processing (1999)
  - Programmable hardware pixel shaders (2001)
  - Hardware floating-point pixel processing (2003)

Graphics Card Performance



Triangle Setup
Projection, lighting, clipping, ...

Pixel Rendering
Texturing, blending

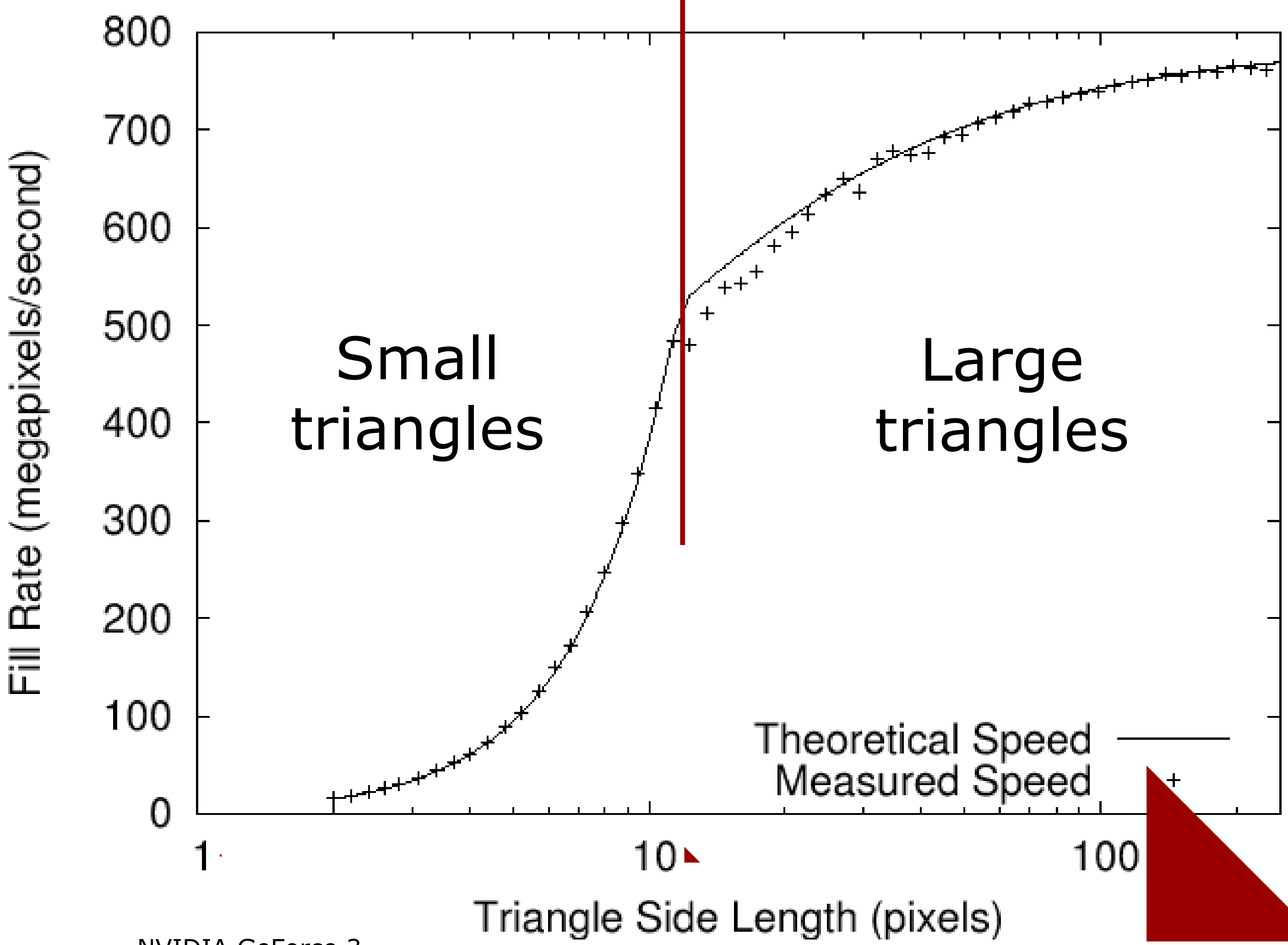
$$t = \max(\alpha, \beta(s + \gamma r))$$

t	total time to draw (seconds)
α	triangle setup time (about 100ns), 1.0/triangle rate
β	pixel rendering time (about 2ns), 1.0/fill rate
s	area of triangle (pixels)
r	rows in triangle
γ	pixel cost per row (about 3 pixels/row)

!

Graphics Card: Usable Fill Rate

$$B_C = \frac{s}{t} = \min(\frac{s}{\alpha}, \frac{1}{\beta(1+\gamma r/s)})$$



NVIDIA GeForce 3

Impostors Technique

- For efficient rendering, must use large triangles; for more detailed rendering, must use smaller triangles
- Impostors can resolve this conflict:
  - First, render set of small triangles into a large texture: an <u>impostor</u>
  - Now we can render impostor texture (on a large triangle) instead of the many small triangles
- Helps when impostors can be <u>reused</u> across many frames
  - Works best with continuous camera motion and high framerate!
- Many modifications, much prior work:

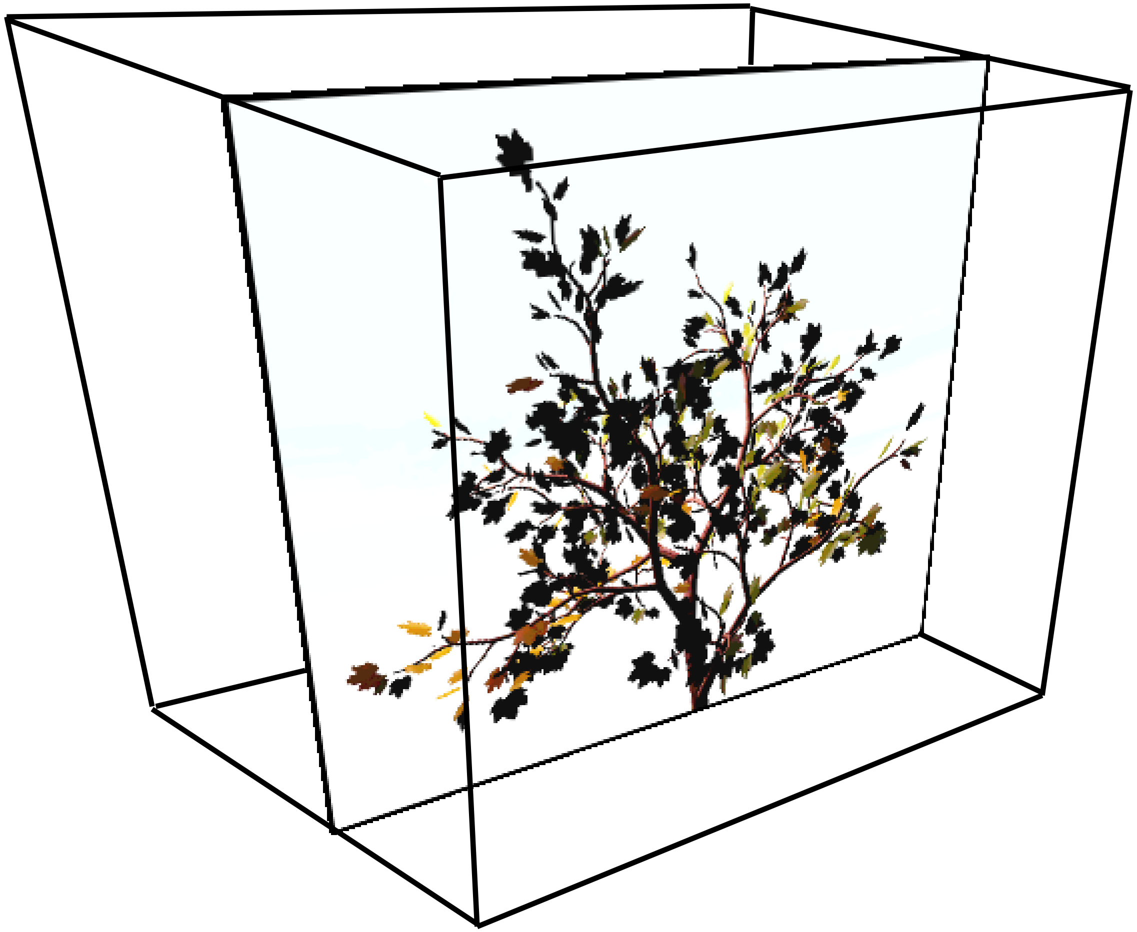
[Maciel95], [Shade96], [Schaufler96]

Impostors: Example

- **We render a set of geometry into an impostor (image/texture)**

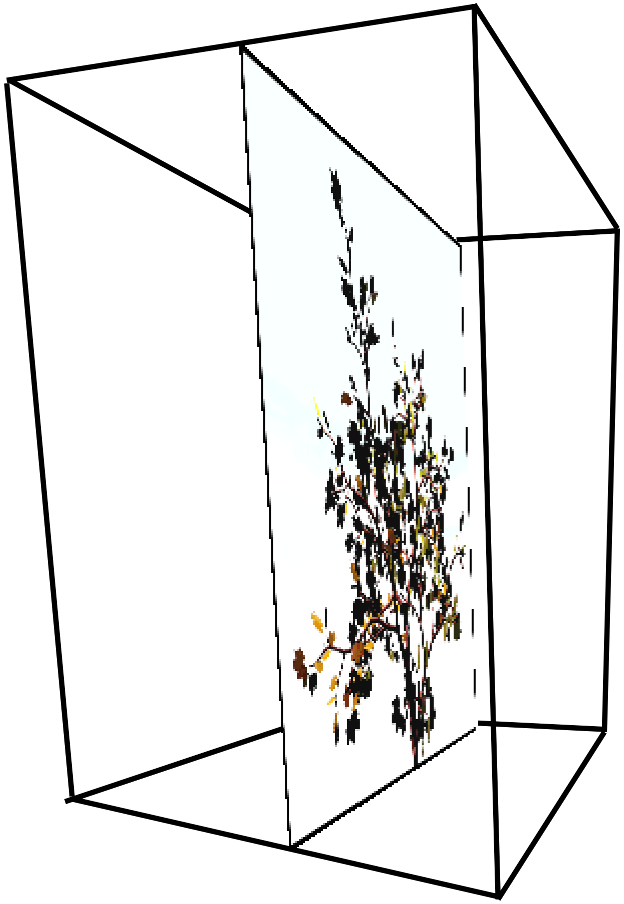
Impostors: Example

- **We can re-use this impostor in 3D for several frames**



Impostors : Example

- **Eventually, we have to update the impostor**



Updating: Impostor Reuse

- **Far away or flat impostors can be reused many times, so impostors help substantially**

	$d = 0.05$	$d = 0.25$	$d = 1$	$d = 5$
$z = 1$	1	1	1	1
$z = 5$	10	2	1	1
$z = 25$	263	52	12	2
$z = 100$	4216	841	208	40

$$R = \frac{z(z-d)\Delta s H}{kdV}$$

R	Number of frames of guaranteed reuse
z	Distance to impostor (meters)
d	Depth flattened from impostor (meters)
Δs	Acceptable screen-space error (1 pixel)
H	Framerate (60 Hz)
k	Screen resolution (1024 pixels across)
V	Camera velocity (20 kmph)

Impostors Challenges

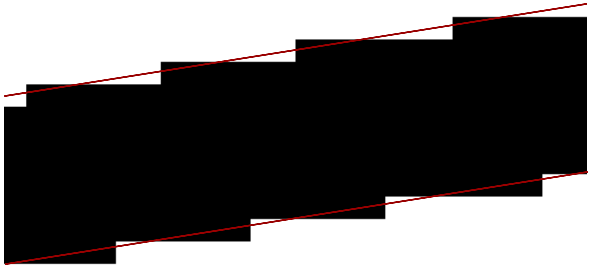
- **Geometry Decomposition**
  - **Must be able to cut up world into impostor-type pieces**
    - **[Shade96] based on scene hierarchy**
    - **[Aliaga99] gives automatic portal method**
  - **Update equation tells us to cut world into flat (small d) pieces for maximum reuse**
- **Update equation shows reuse is low for nearby geometry**
  - **Impostors don’t help much nearby**
  - **Use regular polygon rendering up close**
- **Lots of other reasons for updating:**
  - **Changing object shape, like swaying trees**
  - **Non-diffuse appearance, like reflections**

Impostors Research

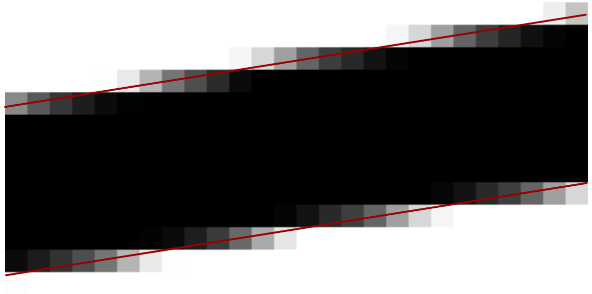
Antialiasing

Motion Blur

Rendering Quality: Antialiasing



Aliased point samples



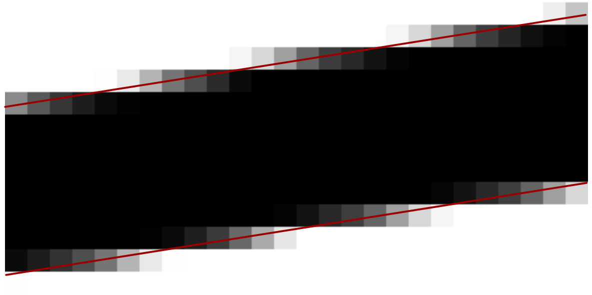
Antialiased filtering

- Real objects can cover only part of a pixel
  - Blends object boundaries
- Prior Work:
  - Ignore partial coverage
    - Aliasing ("the jaggies")
  - Oversample and average
    - Graphics hardware: FSAA
    - Not theoretically correct; close
  - Random point samples
    - [Cook, Porter, Carpenter 84]
    - Needs a *lot* of samples:

$$\sigma' = \frac{\sigma}{\sqrt{n}}$$

  - Integration
    - Trapezoids
    - Circles [Amanatides 84]
    - Polynomial splines [McCool 95]
    - Procedures [Carr & Hart 99]

Antialiased Impostors



Antialiased Impostor

- Texture map filtering is mature
  - Very fast on graphics hardware
  - Bilinear interpolation for nearby textures
  - Mipmaps for distant textures
  - Anisotropic filtering becoming available
  - Works well with alpha channel transparency

    [Haeberli & Segal 93]

- Impostors let us use texture map filtering on *geometry*
  - Antialiased edges
  - Mipmapped distant geometry
  - Substantial improvement over ordinary polygon rendering

Antialiased Impostor Challenges

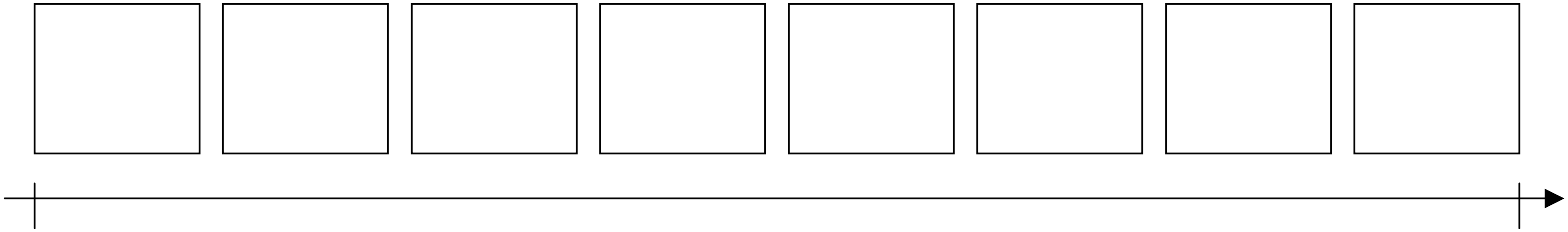
- Must generate antialiased impostors to start with
  - Just pushes antialiasing up one level
  - Can use any antialiasing technique. We use:
    - Trapezoid-based integration
    - Blended splats

- Must render with transparency
  - Not compatible with Z-buffer
  - Painter’s algorithm:
    - Draw from back-to-front
    - A radix sort works well
    - For terrain, can avoid sort by traversing terrain properly

Rendering Quality: Motion Blur

- **Fast-moving objects blur**
- **Prior Work (as before)**
  - **Just temporal aliasing**
- **Usual method**
  - **Draw geometry shifted to different times**
  - **One shift per pixel of blur distance**
  - **Average shifted images together using accumulation buffer**
- **New Idea: fast exponentiation blur**
  - **Draw geometry *once***
  - **Read back, shift, repeat**
  - **No accumulation buffer needed**

Normal Motion Blur

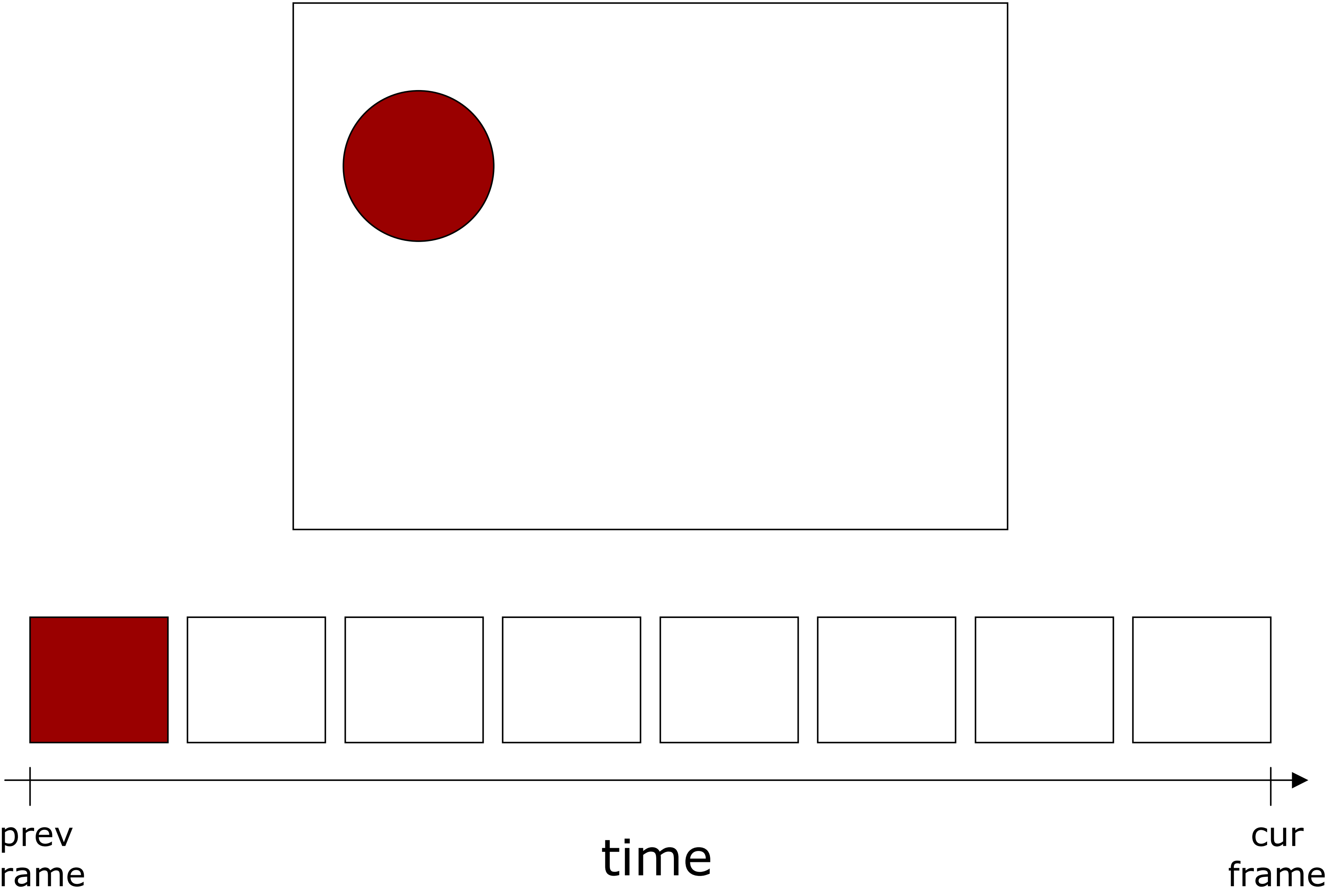


prev
frame

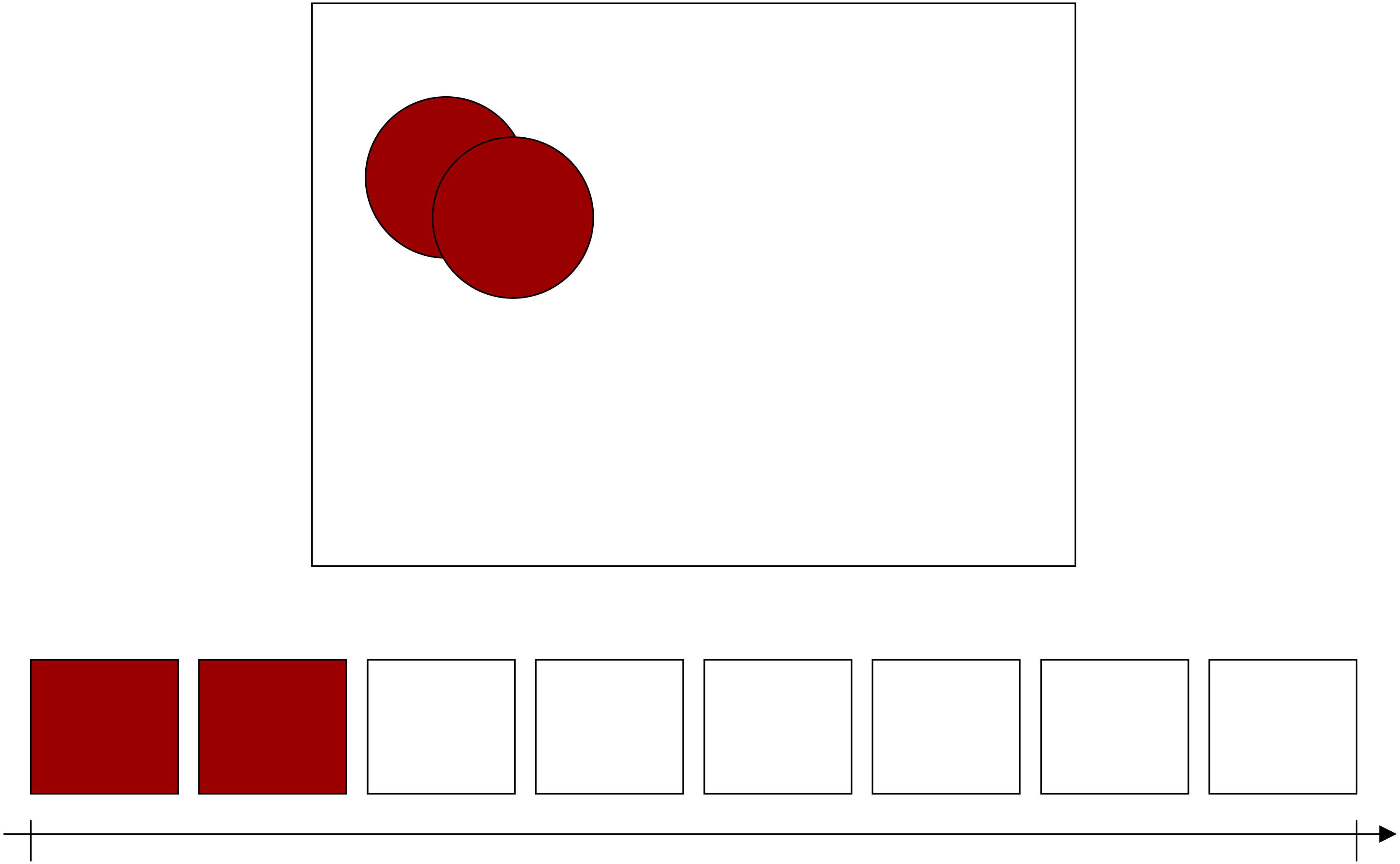
time

cur
frame

Normal Motion Blur



Normal Motion Blur

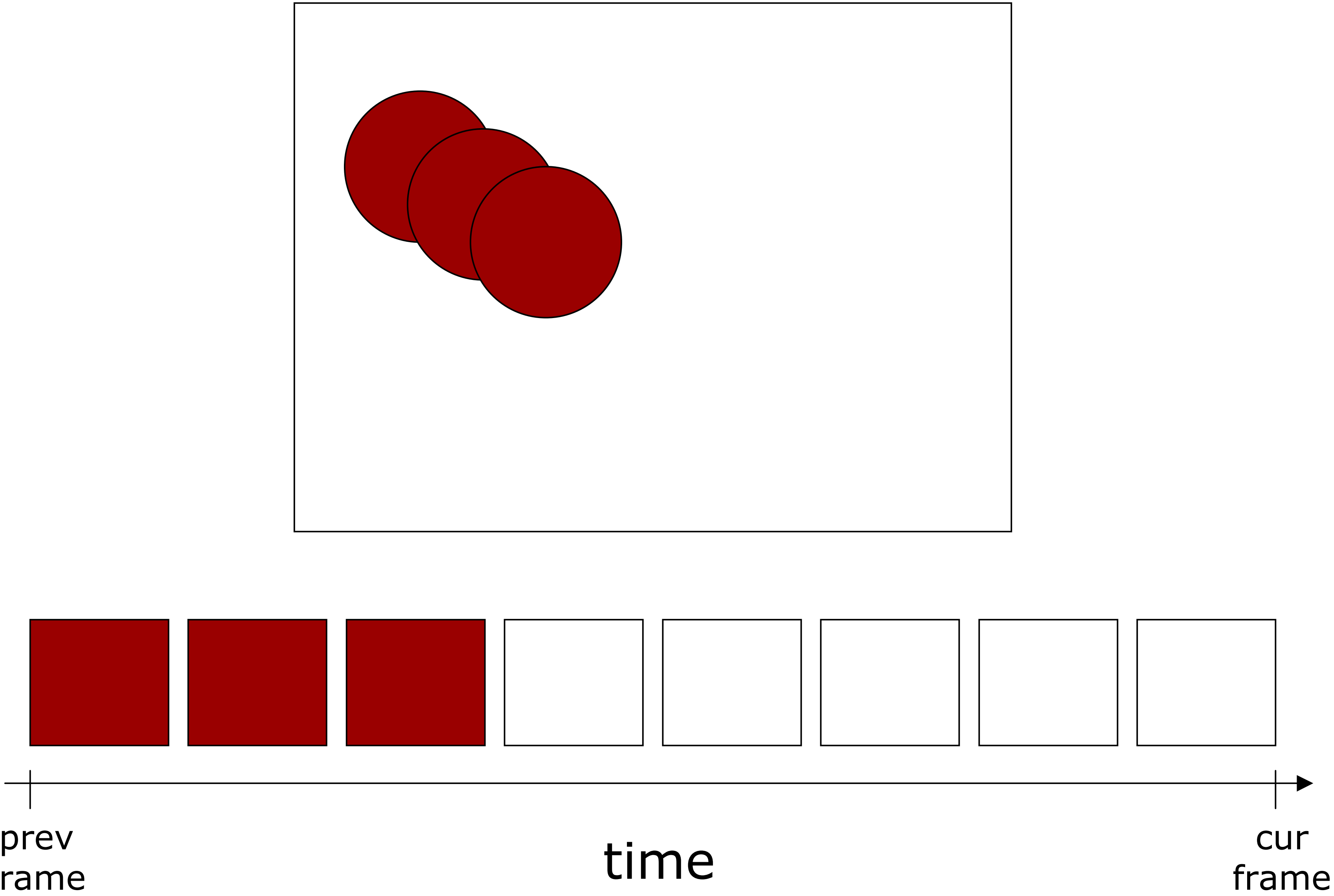


prev
frame

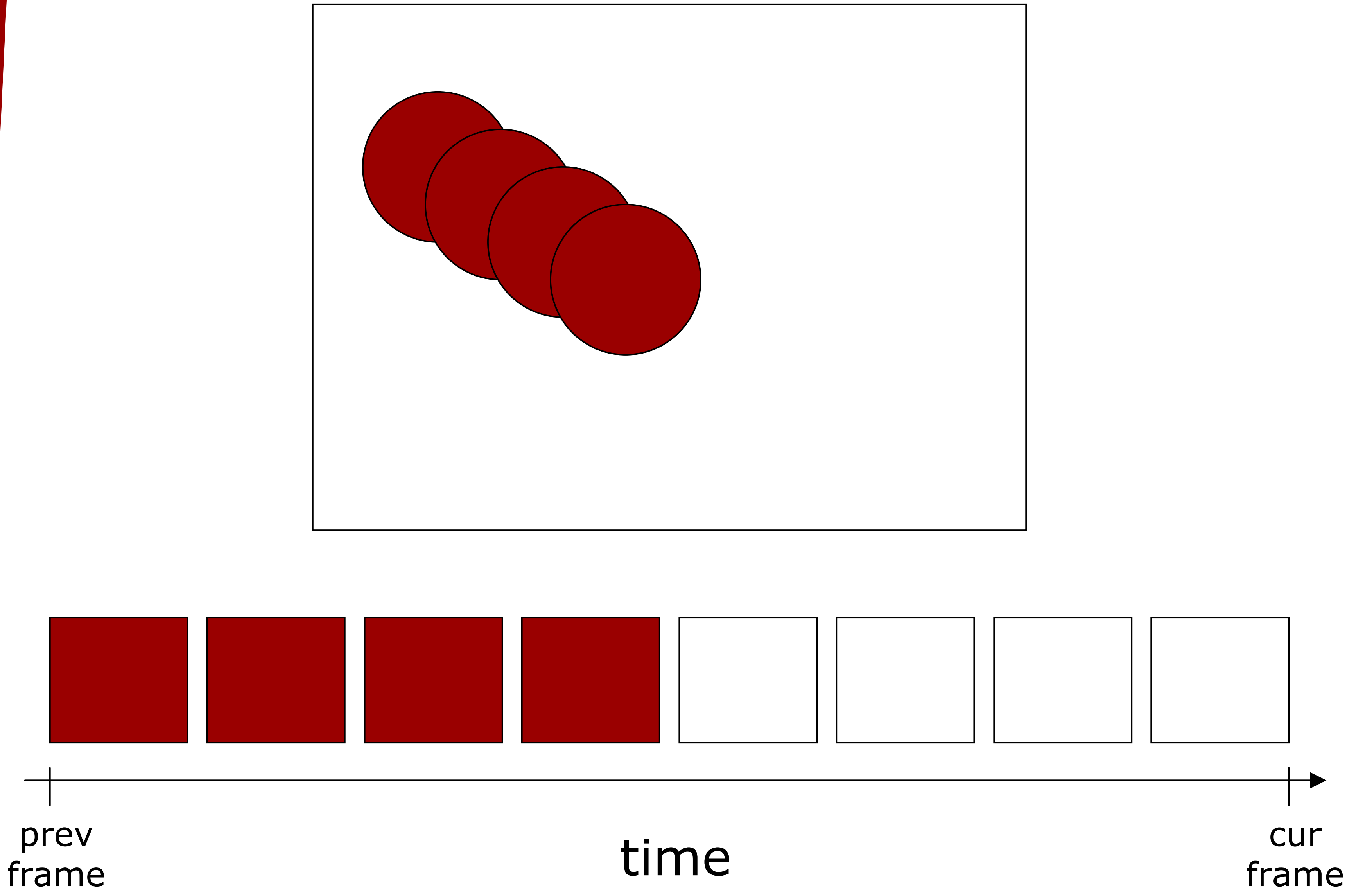
time

cur
frame

Normal Motion Blur



Normal Motion Blur

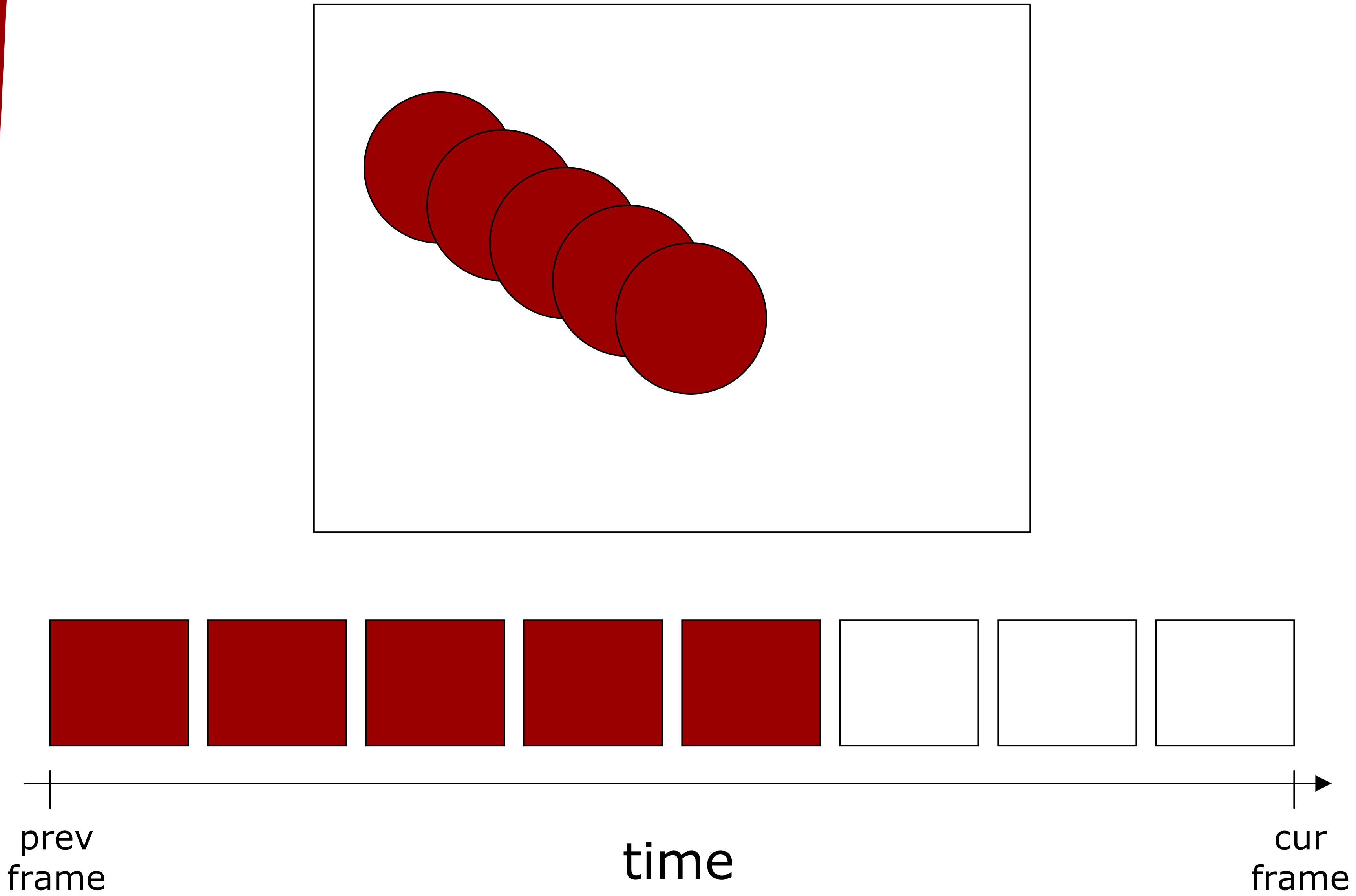


prev
frame

time

cur
frame

Normal Motion Blur

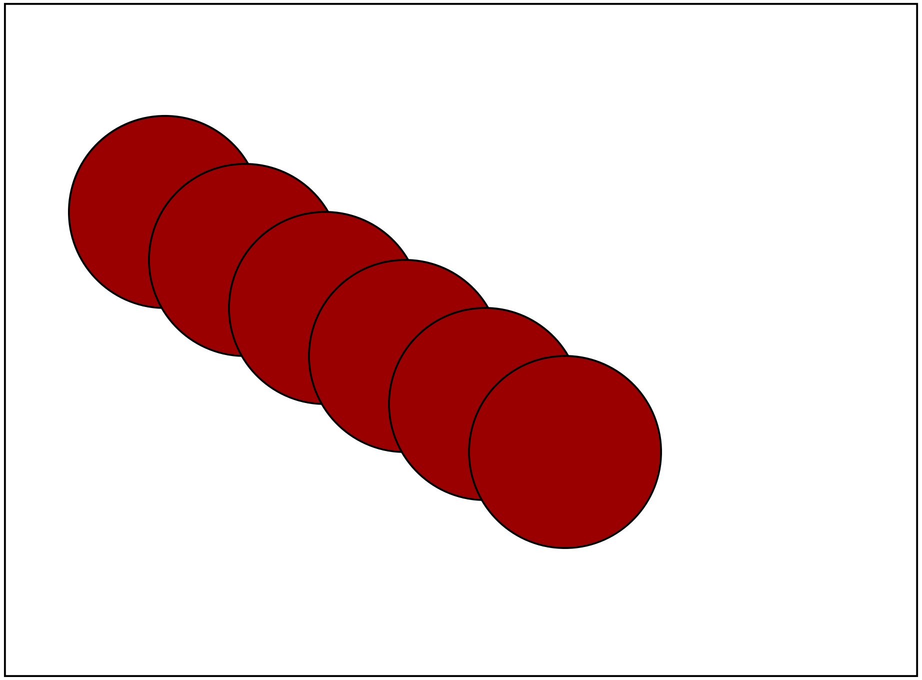


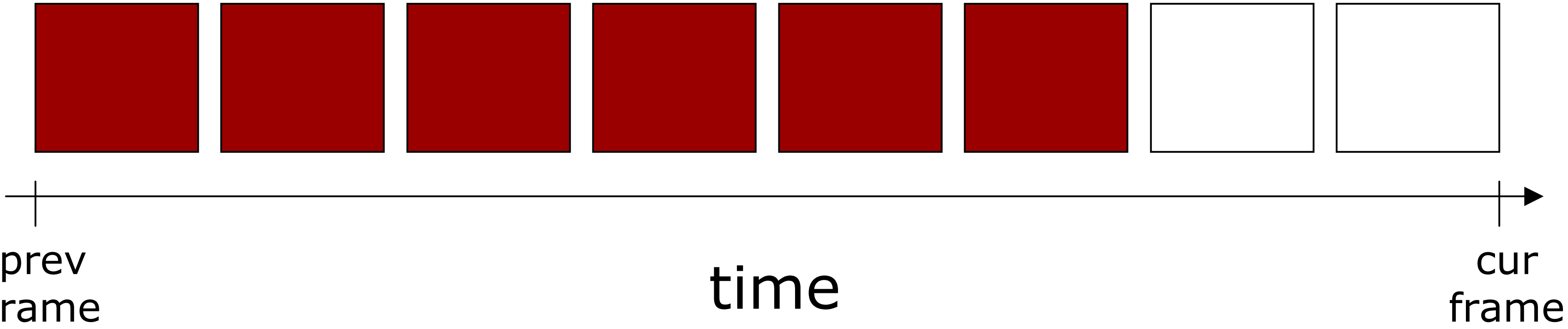
prev
frame

time

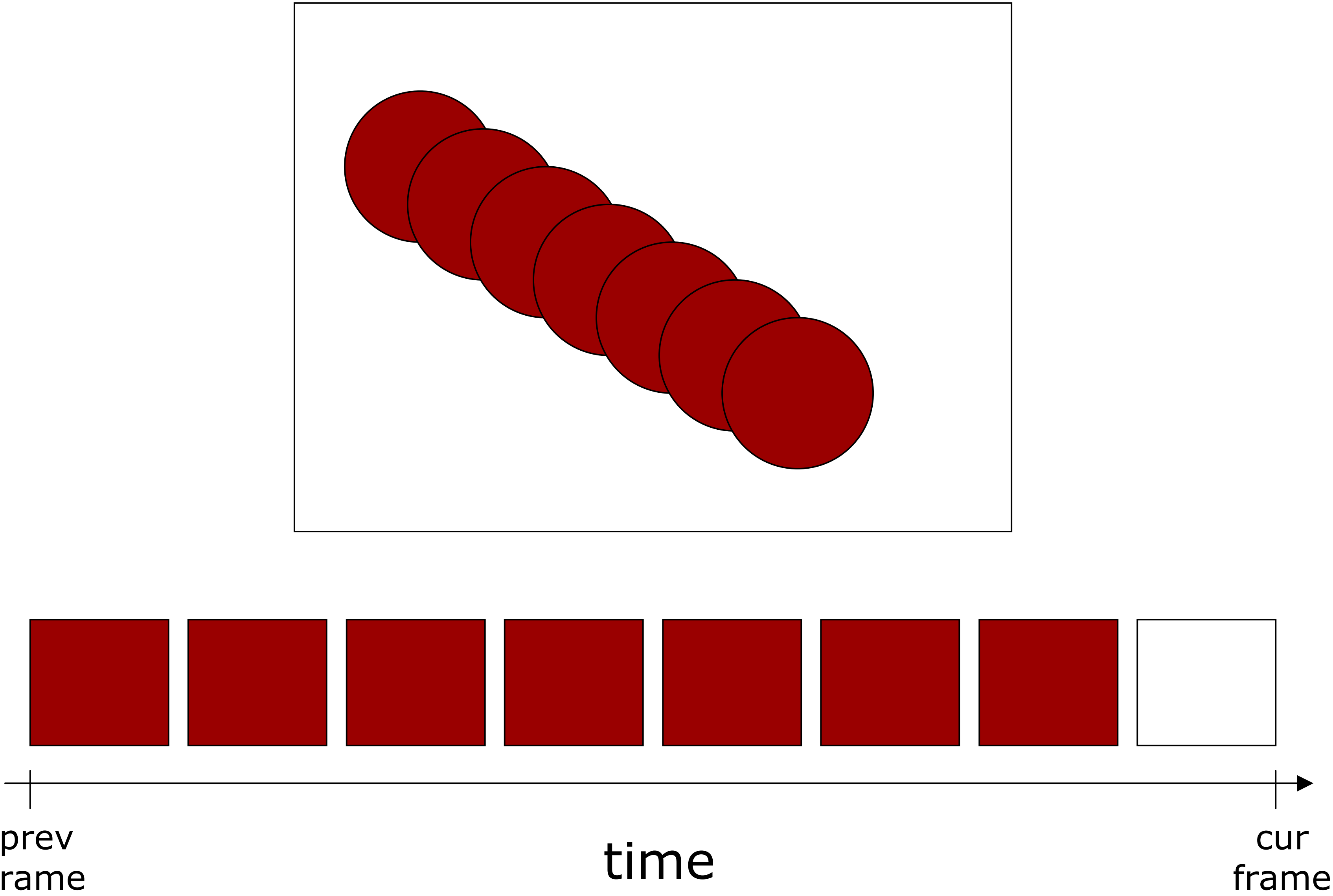
cur
frame

Normal Motion Blur

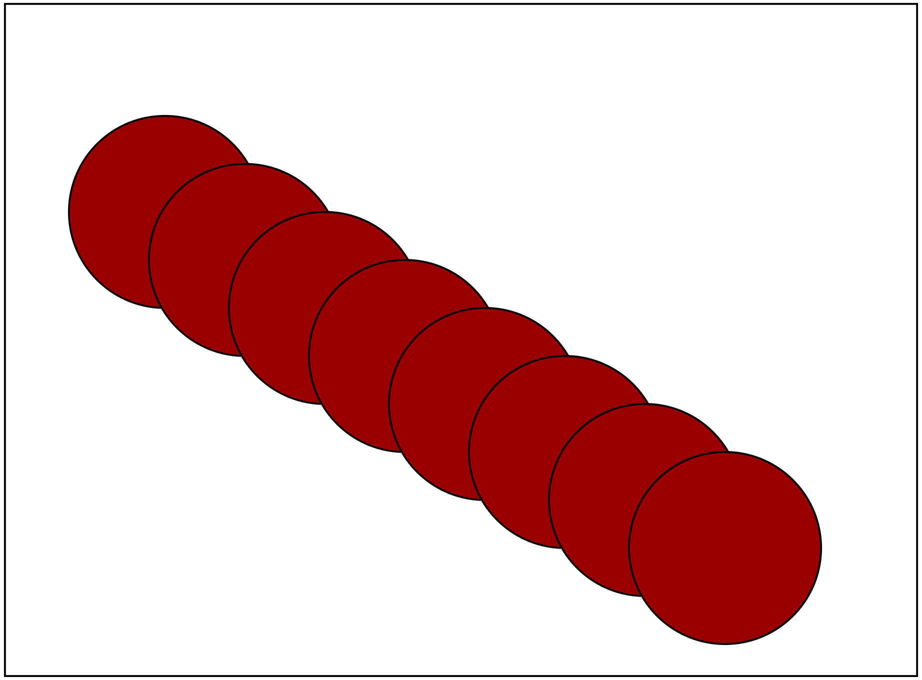




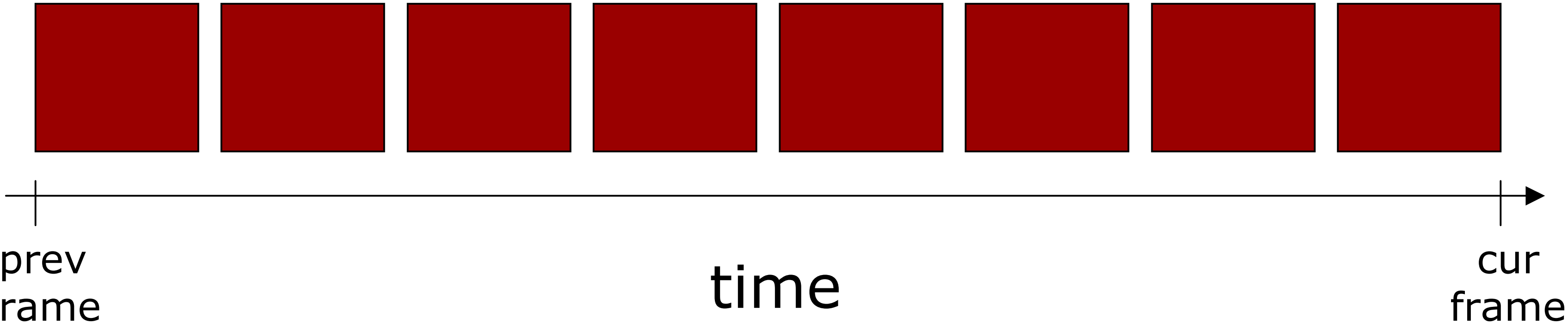
Normal Motion Blur



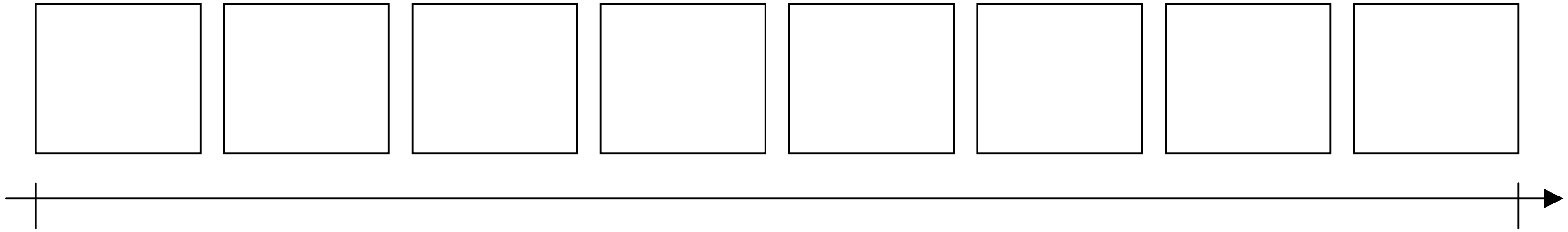
Normal Motion Blur



n shifts take $O(n)$ time



Fast Exponentiation Blur

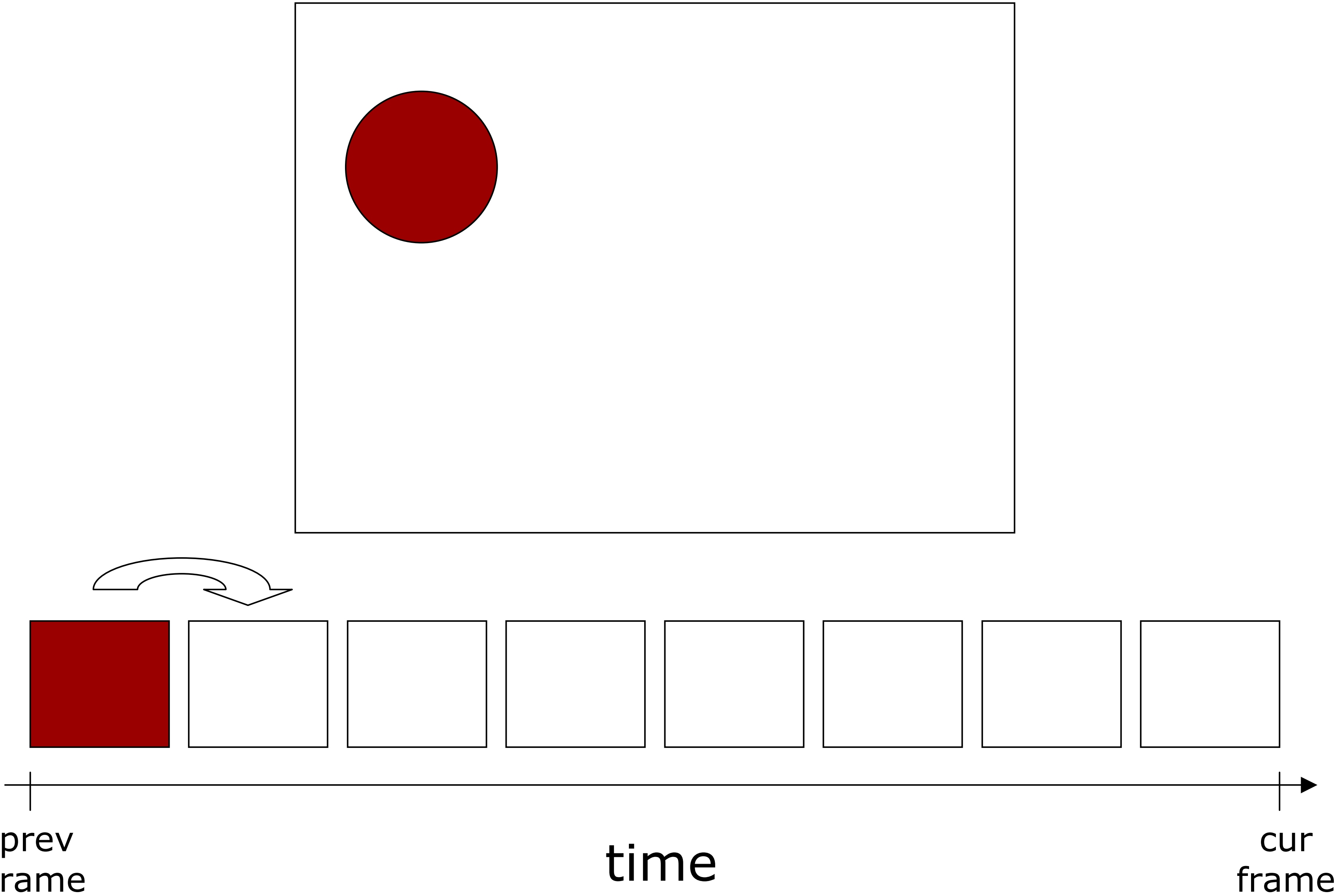


prev
frame

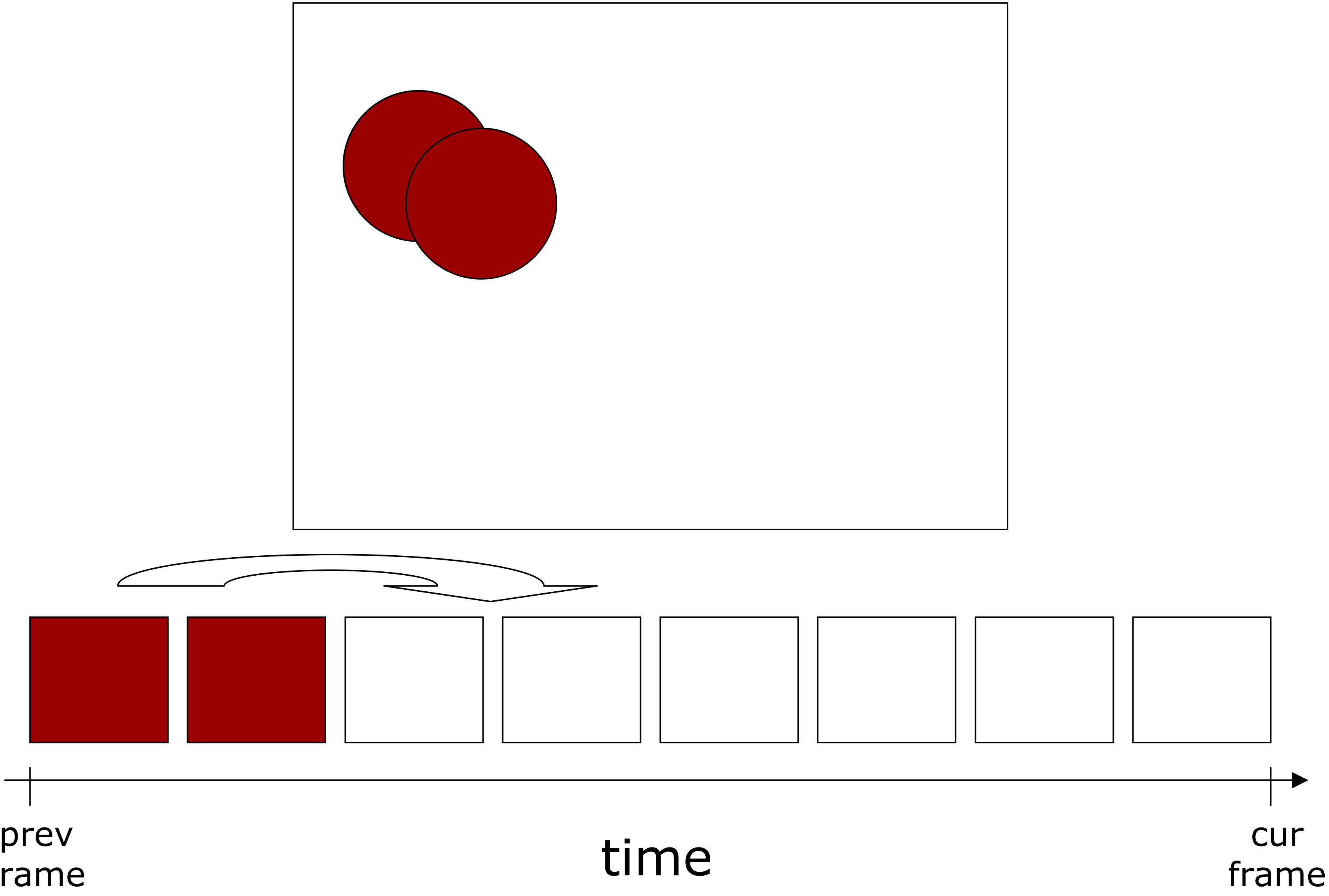
time

cur
frame

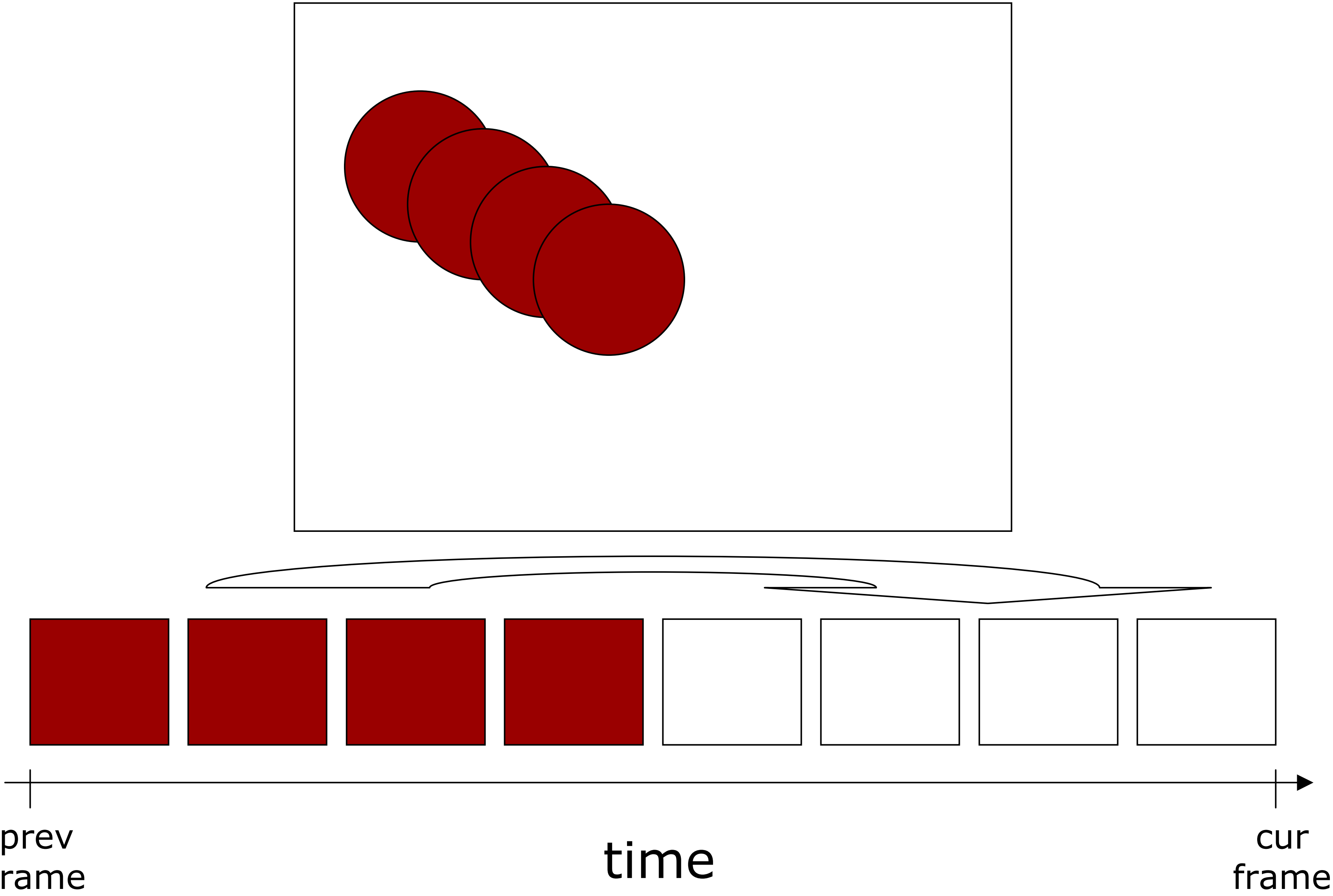
Fast Exponentiation Blur



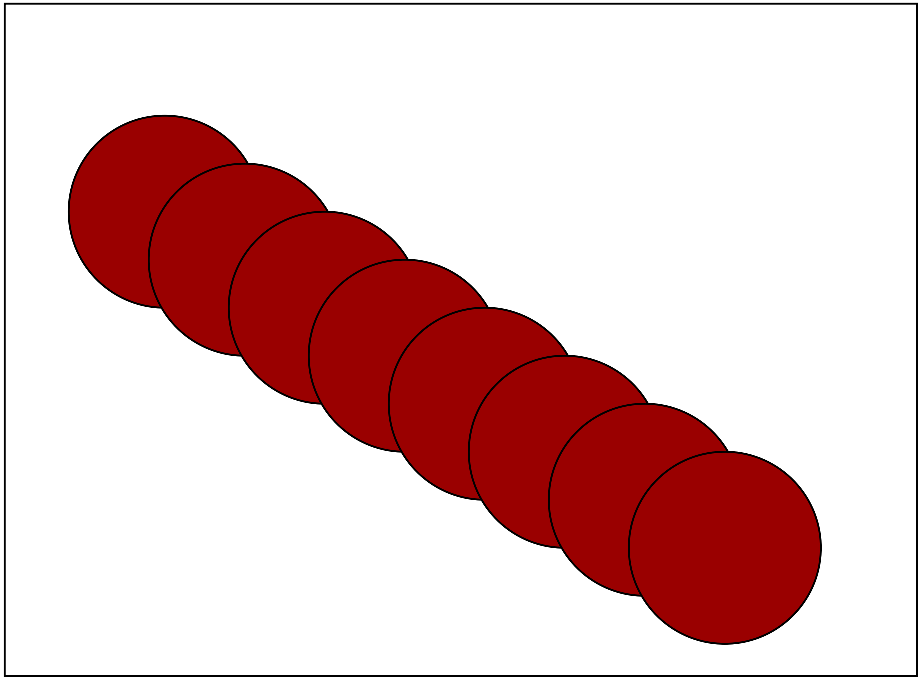
Fast Exponentiation Blur



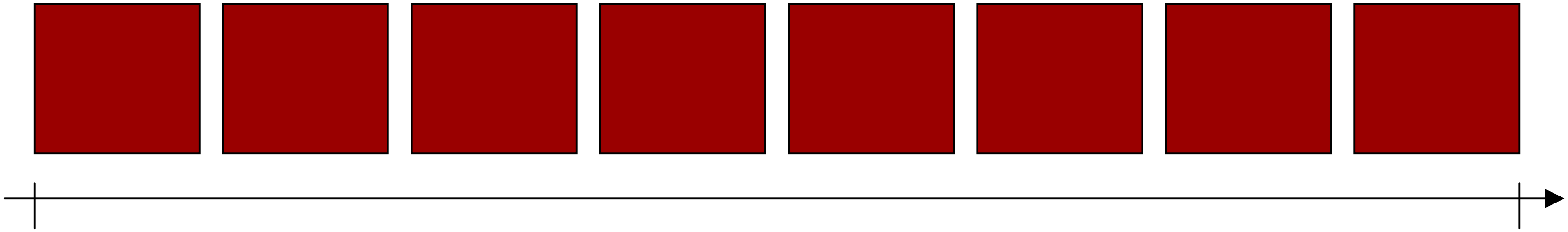
Fast Exponentiation Blur



Fast Exponentiation Blur



n shifts take $O(\lg n)$ time



prev frame

time

cur frame

Impostors Research Summary

- Impostors can improve the rendering quality, not just speed
  - Antialiasing
  - Motion Blur
- This is possible because impostors let you process *geometry* like a *texture*
  - Filtering for antialiasing
  - Repeated readback for motion blur

Parallel Rendering

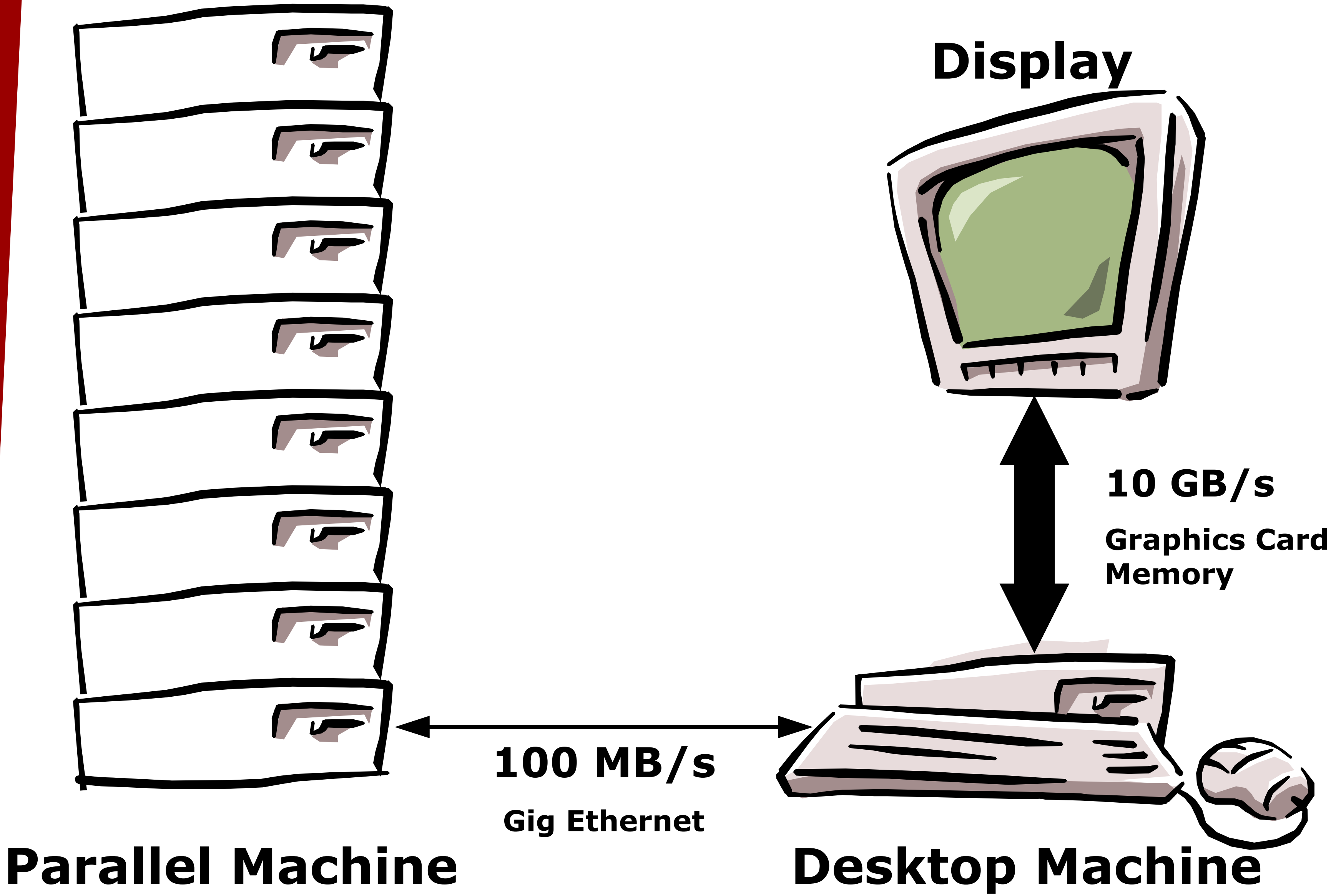
Fundamentals
Prior Work

Parallel Rendering

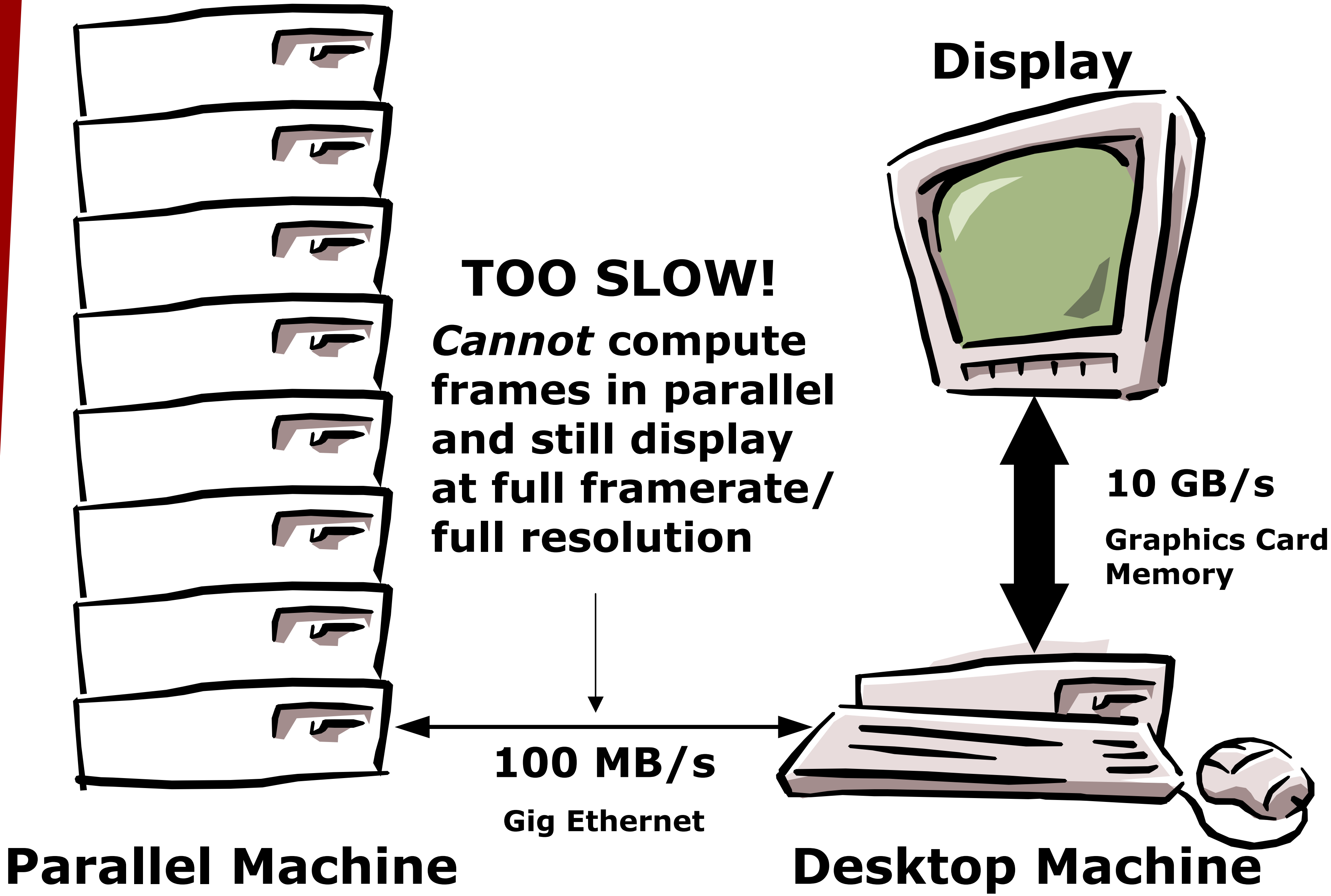
- Huge amounts of prior work in offline rendering
  - Non-interactive: no human in the loop
  - Not bound by framerate: can take seconds to hours
- Tons of raytracers [John Stone’s Tachyon], radiosity solvers [Stuttard 95], volume visualization [Lacroute 96], etc
- “Write an MPI raytracer” is a homework assignment
- Movie visual effects studios use frame-parallel offline rendering (“render farm”)

- Basically a solved problem

Interactive Parallel Rendering



Interactive Parallel Rendering



Display

TOO SLOW!

Cannot **compute frames in parallel and still display at full framerate/ full resolution**

10 GB/s

Graphics Card Memory

100 MB/s

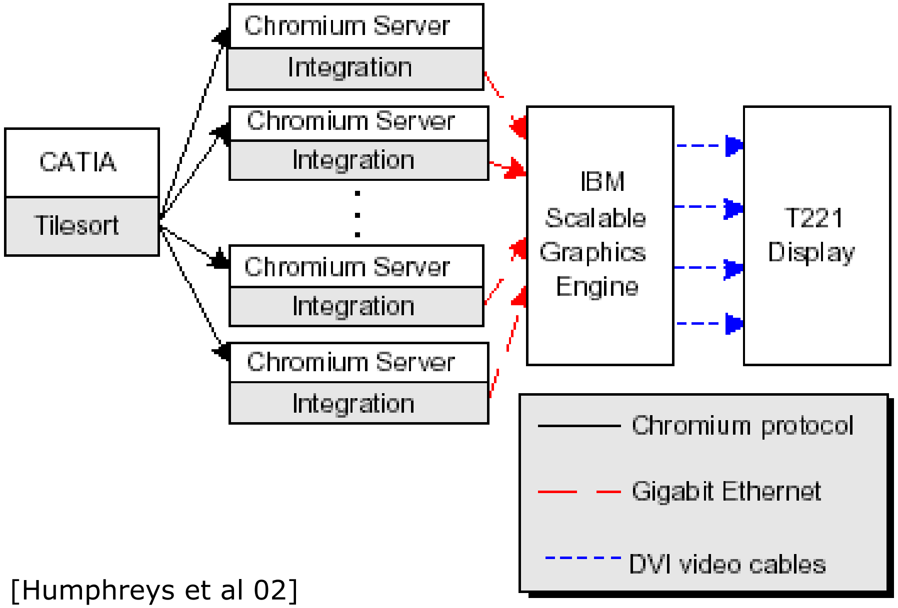
Gig Ethernet

Parallel Machine

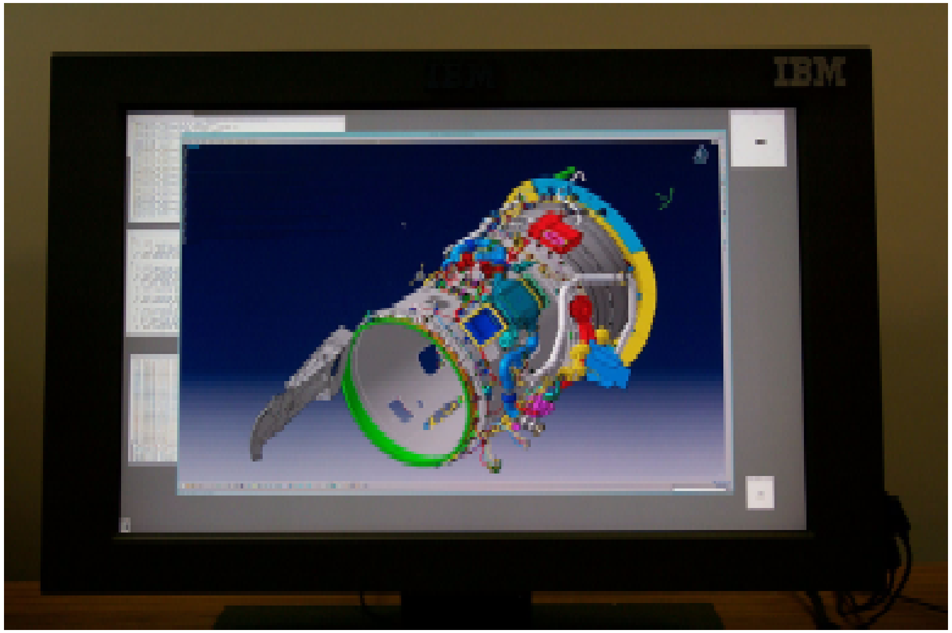
Desktop Machine

Interactive Parallel Rendering

- Humphreys et al's Chromium (aka Stanford's WireGL)
  - Binary-compatible OpenGL shared library
  - Routes OpenGL commands across processors efficiently
  - Flexible routing--arbitrary processing possible
  - Typical usage: parallel geometry generation, screen-space divided parallel rendering
  - Big limitation: screen image reassembly bandwidth
    - Multi-pipe custom image assembly hardware on front end

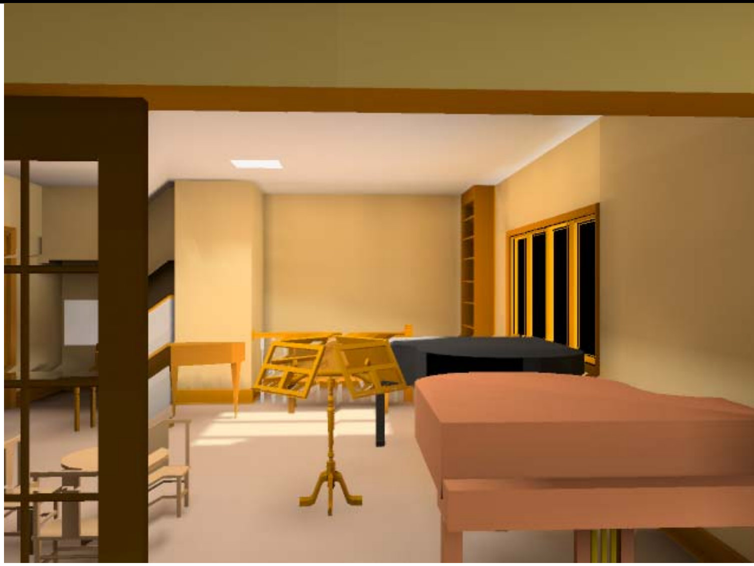


[Humphreys et al 02]



Interactive Parallel Rendering

- Bill Mark's post-render warping
  - Parallel server sends every N'th frame to client
  - Client interpolates remaining frames by warping server frames according to depth



[Mark 99]

[Ward 99]

- Greg Ward's "ray cache"
  - Parallel Radiance server renders and sends bundles of rays to client
  - Client interpolates available nearby rays to form image

Parallel Impostors

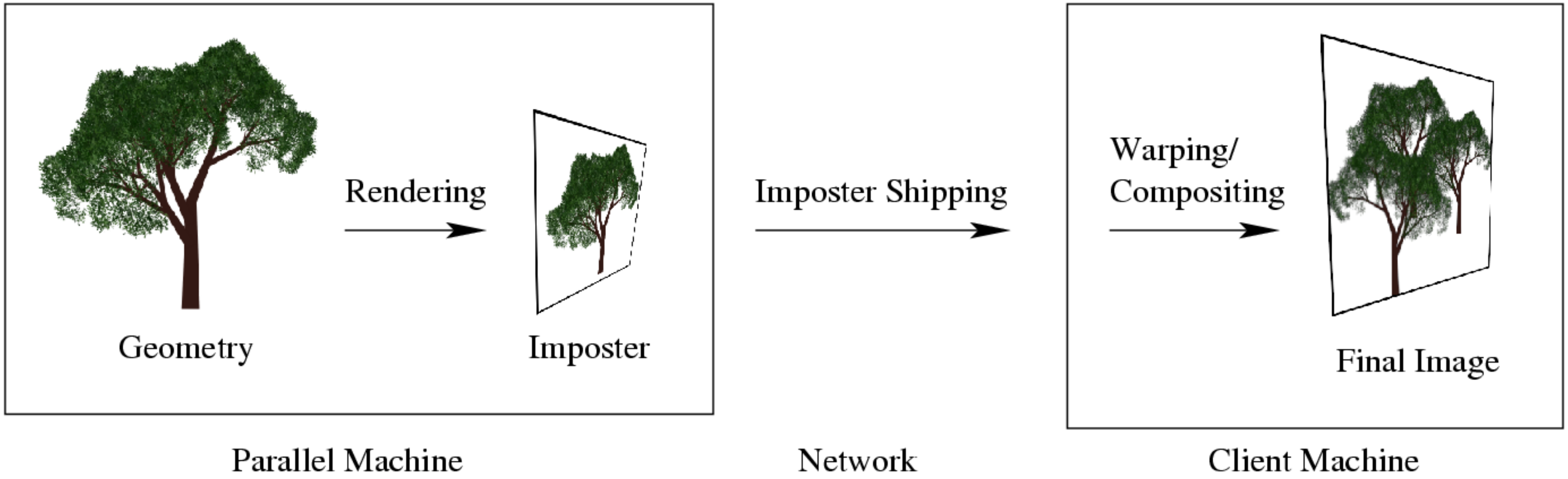
Our Main Technique

Parallel Impostors Technique

- Render pieces of geometry into impostor images on parallel server
  - Parallelism is across impostors
    - Fine grained-- lots of potential parallelism
    - Geometry is partitioned by impostors anyway
  - Reassemble world on serial client
    - Uses rendering bandwidth of graphics card

- Impostor reuse cuts required network bandwidth to client
  - Only update images when necessary
- Uses the speed and memory of the parallel machine

Client/Server Architecture

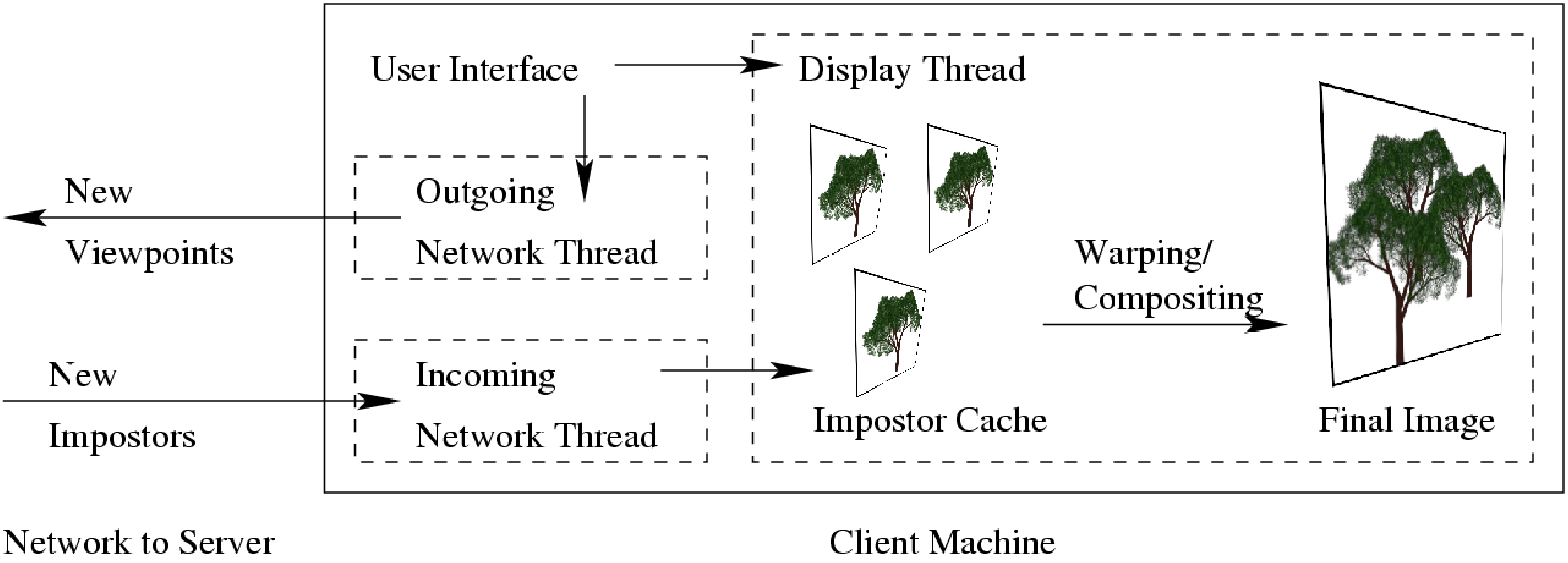


- Client sits on user's desk
  - Sends server new viewpoints
  - Receives and displays new impostors
- Server can be anywhere on network
  - Renders and ships back new impostors as needed

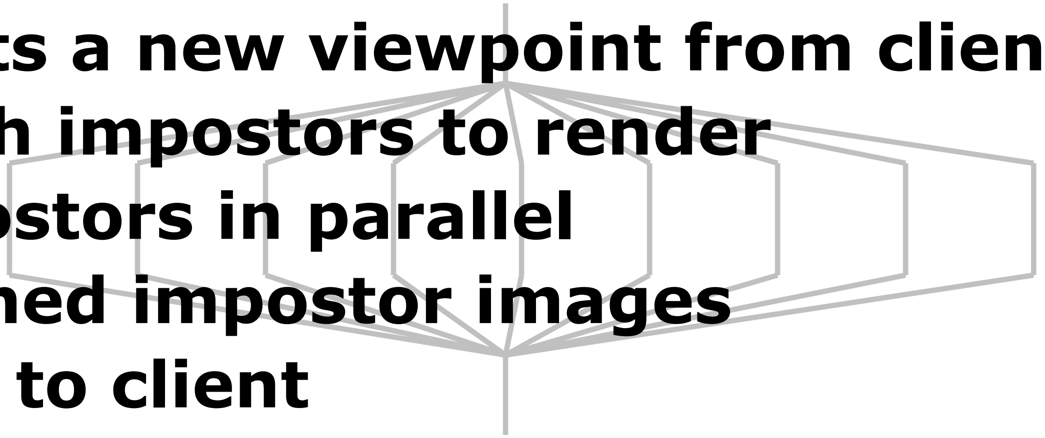
- Implementation uses TCP/IP sockets
  - CCS & PUP protocol [Jyothi and Lawlor 04]
  - Works over NAT/firewalled networks

Client Architecture

- Client should *never* wait for server
  - Display existing impostors at fixed framerate
    - Even if they're out of date
  - Prefers spatial error (due to out of date impostor) to temporal error (due to dropped frames)
- Implementation uses OpenGL, kernel threads



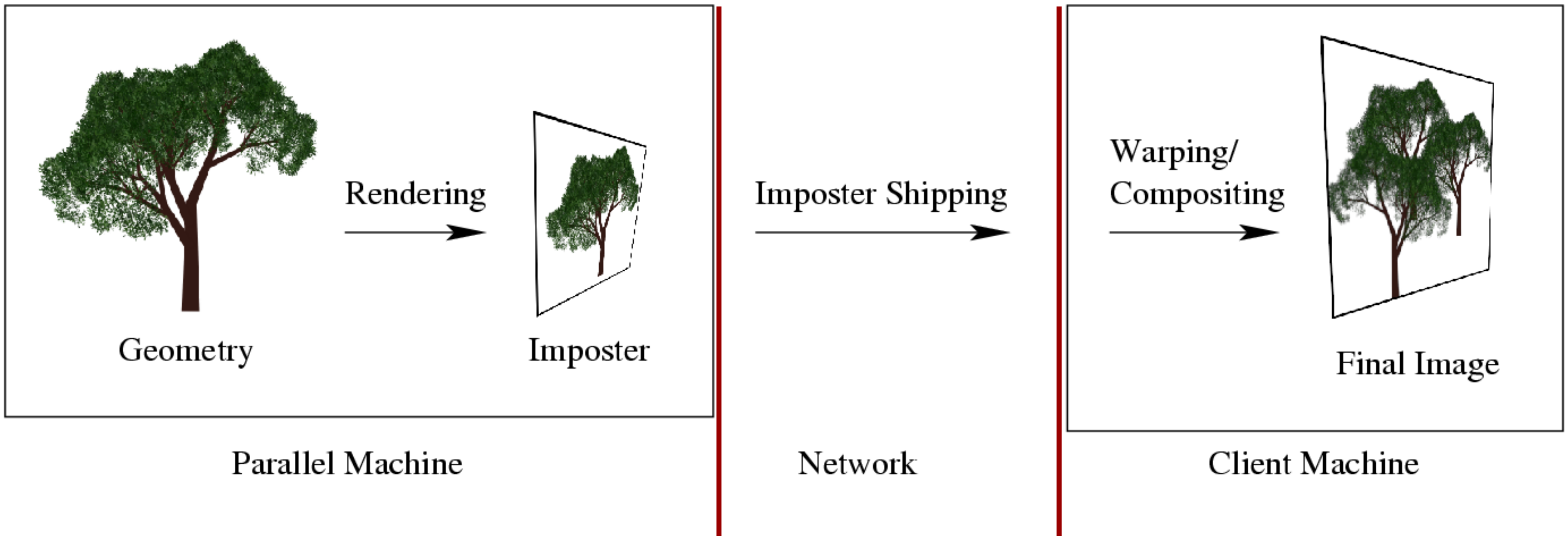
Server Architecture



- Server accepts a new viewpoint from client
- Decides which impostors to render
- Renders impostors in parallel
- Collects finished impostor images
- Ships images to client

- Implementation uses Charm++ parallel runtime
  - Different phases all run *at once*
    - Overlaps everything, to avoid synchronization
    - Much easier in Charm than in MPI
  - Geometry represented by efficient migrateable objects called <u>array elements</u> [Lawlor and Kale 02]
  - Geometry rendered in priority order
  - Create/destroy array elements as geometry is split/merged

Architecture Analysis



$$B = \min(B_R P R,\ B_N C_N R,\ B_C)$$

Benefit from Parallelism

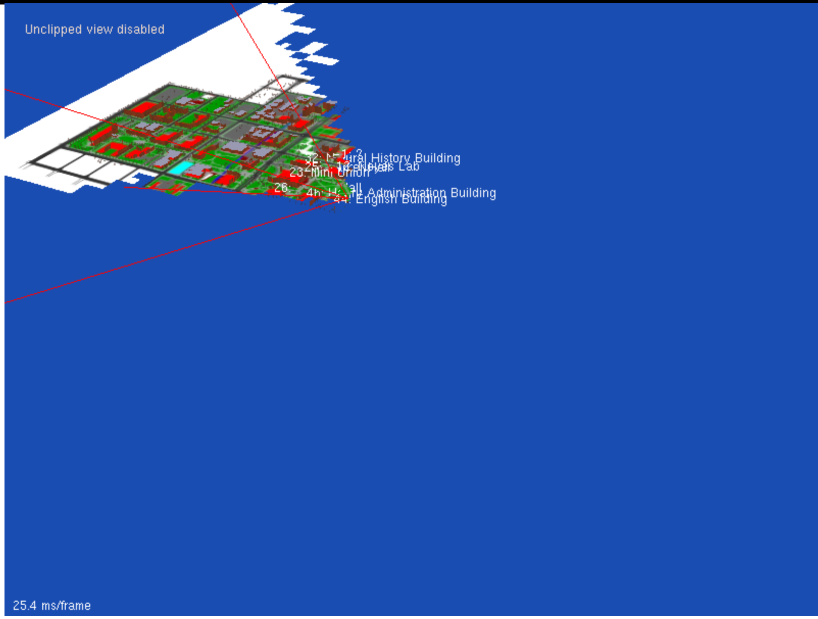
Benefit from Impostors

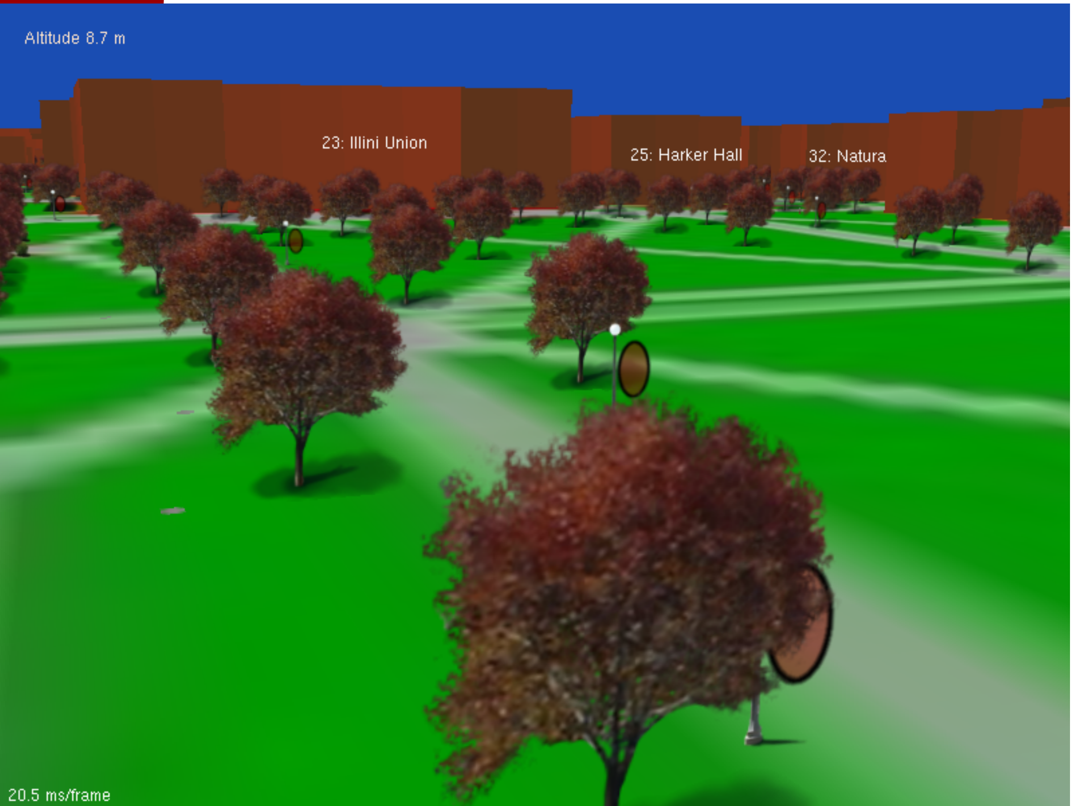
B	Delivered bandwidth (e.g., 300Mpixels/s)
B_R	Rendering bandwidth per processor (e.g., 1Mpixels/s/cpu)
P	Parallel speedup (e.g., 30 effective cpus)
R	Number of frames impostors are reused (e.g., 10 reuses)
B_N	Network bandwidth (e.g., 60 Mbytes/s)
C_N	Network compression rate (e.g., 0.5 pixels/byte)
B_C	Client rendering bandwidth (e.g., 300Mpixels/s)

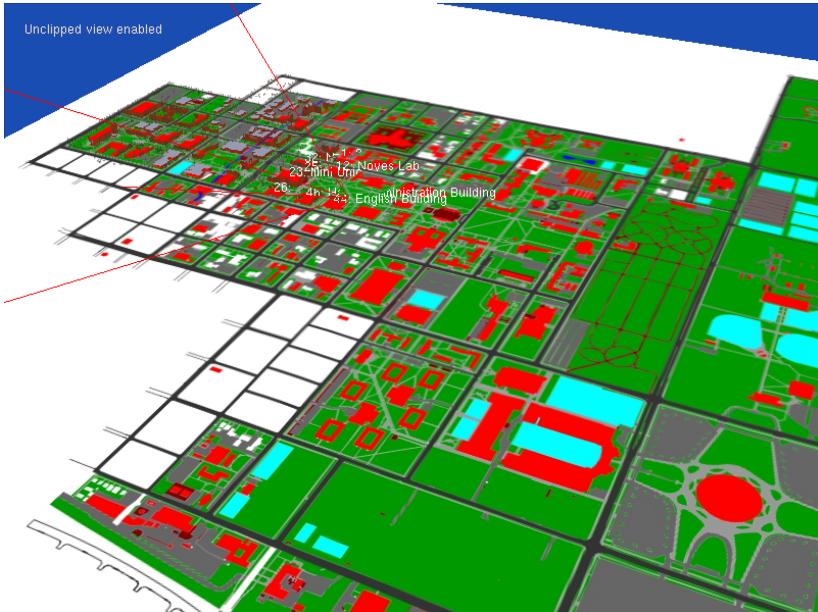
Parallel Planned Work

Complicated, Dynamic Problem

- Only a small fraction of geometry visible & relevant
  - Behind viewer, covered up, too far away…
- Relevant geometry *changes* as camera moves







Prioritized Load Balancing

- Parallelism only provides a benefit *if* problem speedup is good
    - Poor prioritization can destroy speedup
    - Speedup does not mean “all processors are busy”
        - That’s easy, but work must be *relevant*
          [Kale et al 93]
    - Must keep all processors and the network busy on relevant work
- Goal: generate most image improvement for least effort
- Priority for rendering or shipping impostor based on
    - Visible error in the current impostor (pixels)
    - Visible screen area (pixels)
    - Visual/perceptual “importance” (scaling factor)
    - Effort required to render or ship impostor (seconds)
- All of these are estimates!

Graphics Planned Work

New Graphics Opportunities

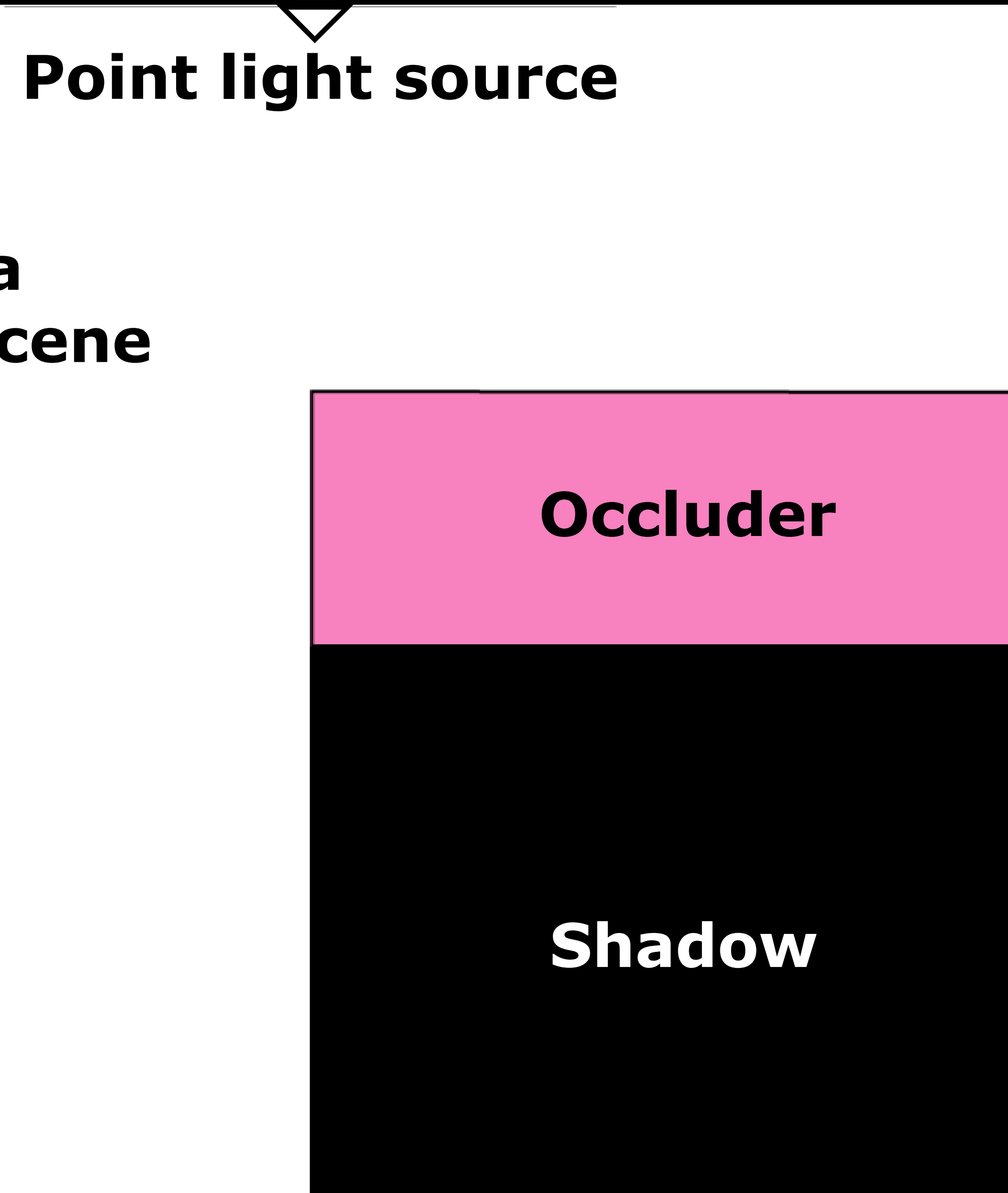
- Impostors cuts the rendering bandwidth needed
- Parallelism provides extra rendering power
- Together, these allow
  - Soft Shadows
  - Global Illumination
  - Procedural Detail Generation
  - Huge models

Quality: Soft Shadows

- Extended light sources cast fuzzy shadows
  - E.g., the sun
- Prior work
  - Ignore fuzziness
  - Point sample area source
  - New faster methods [Hasenfratz 03 survey]

Hard Shadows

Cross section of a hard-shadow scene



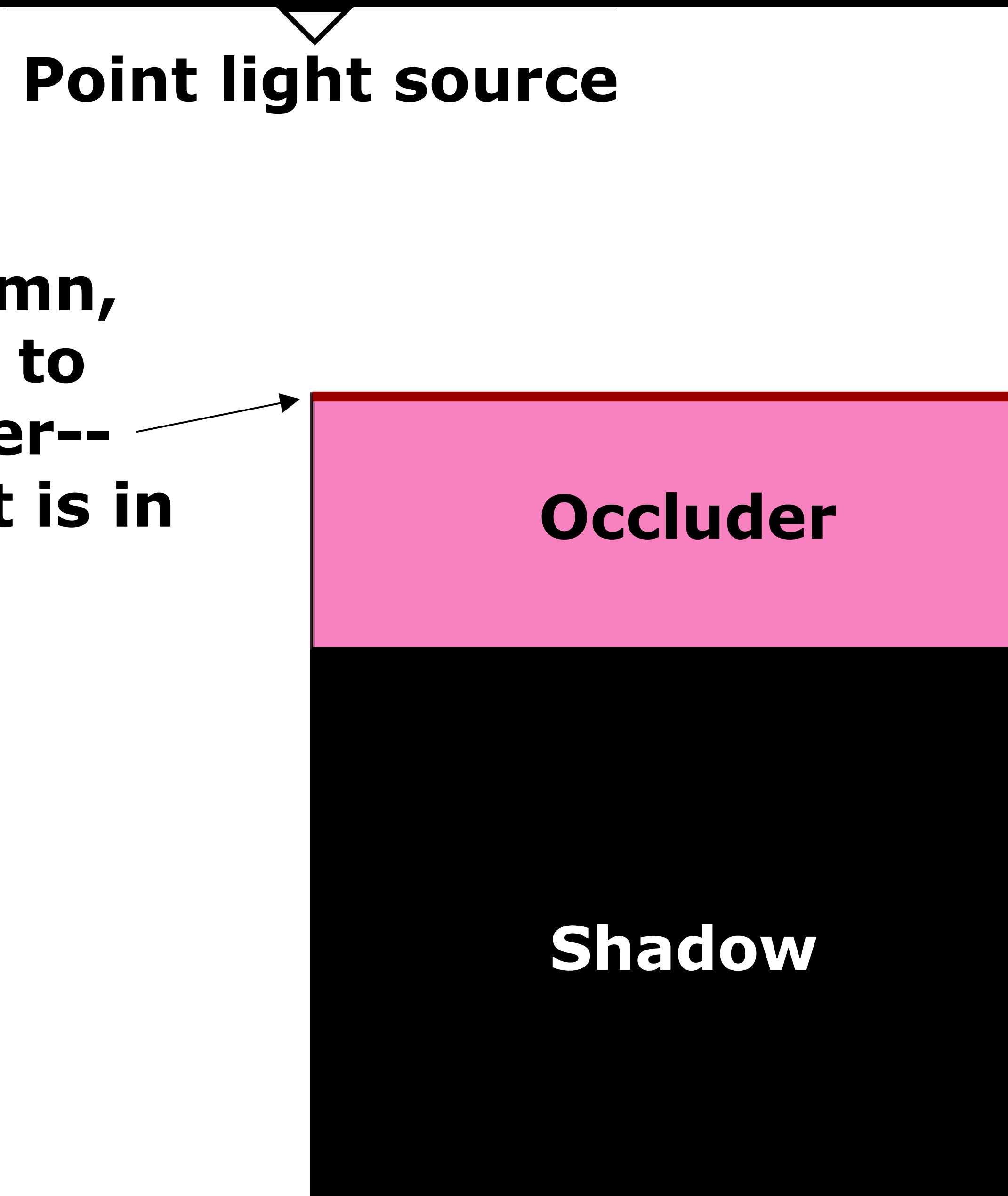
Point light source

Occluder

Shadow

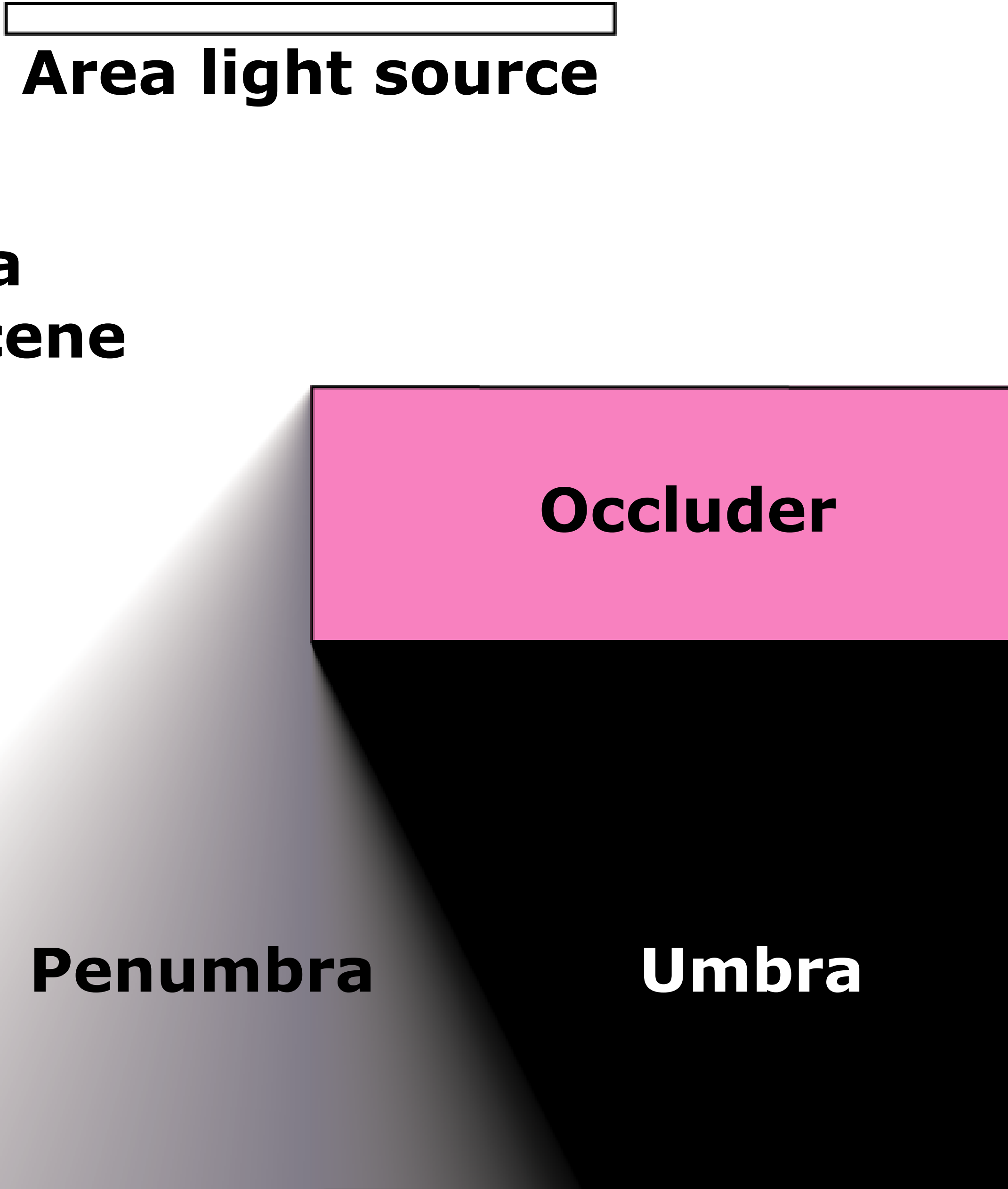
Fully Lit

Hard Shadows: Shadow Map



For each column, store depth to first occluder--beyond that is in shadow

Soft Shadows



Area light source

Cross section of a
soft-shadow scene

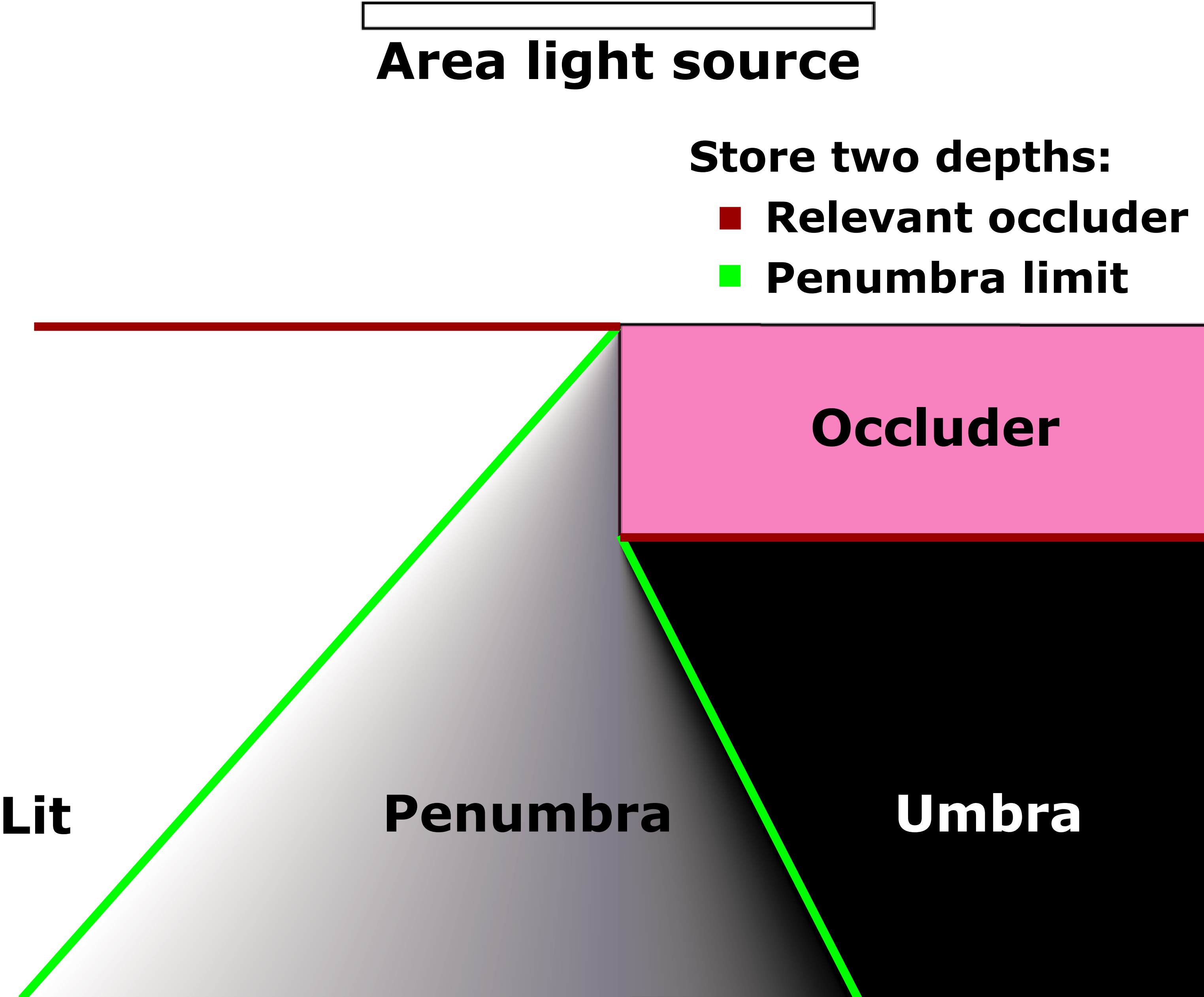
Occluder

Fully Lit

Penumbra

Umbra

Penumbra Limit Map (new)



Area light source

Store two depths:

- 🟥 Relevant occluder
- 🟩 Penumbra limit

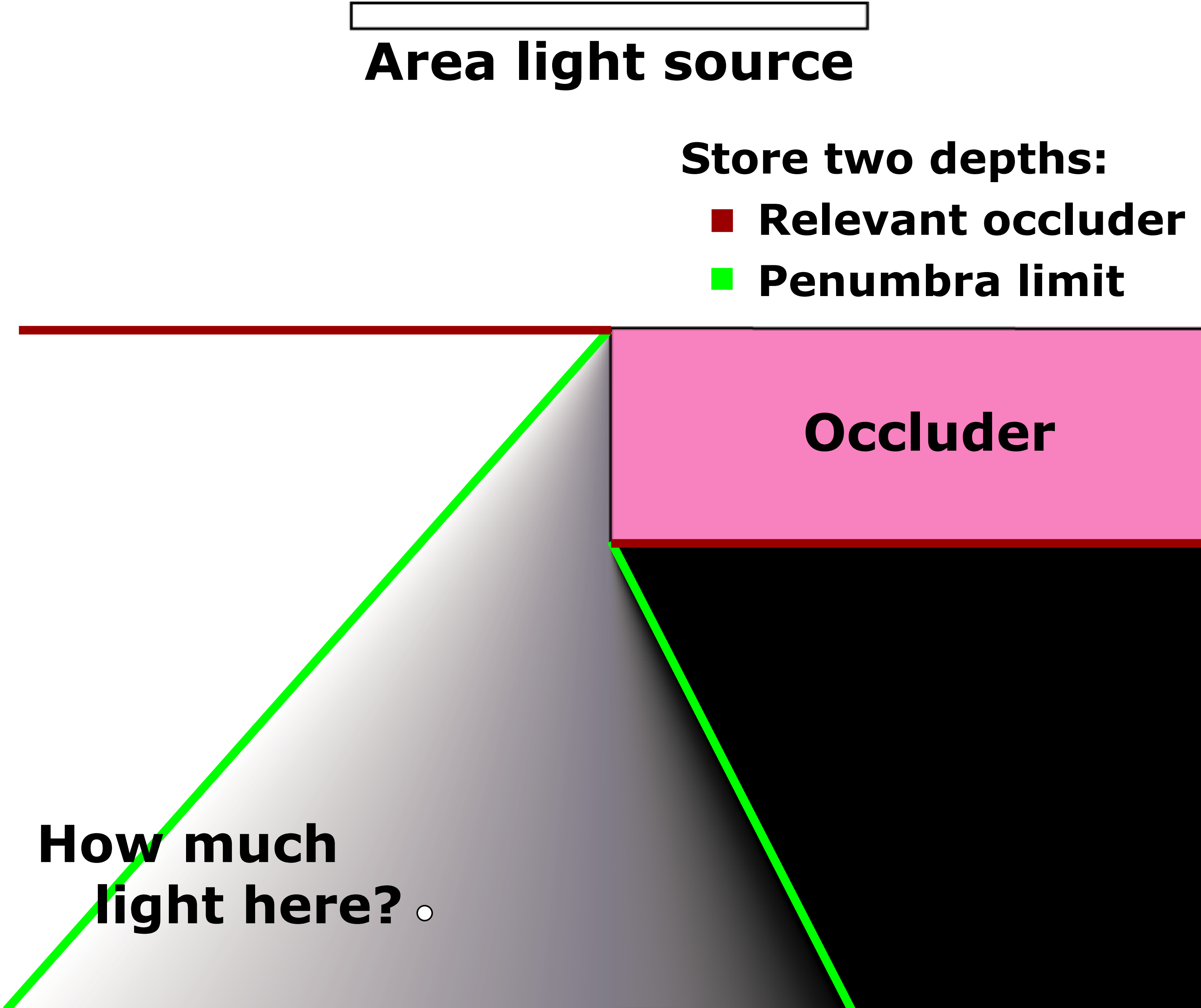
Occluder

Fully Lit

Penumbra

Umbra

Penumbra Limit Map



Area light source

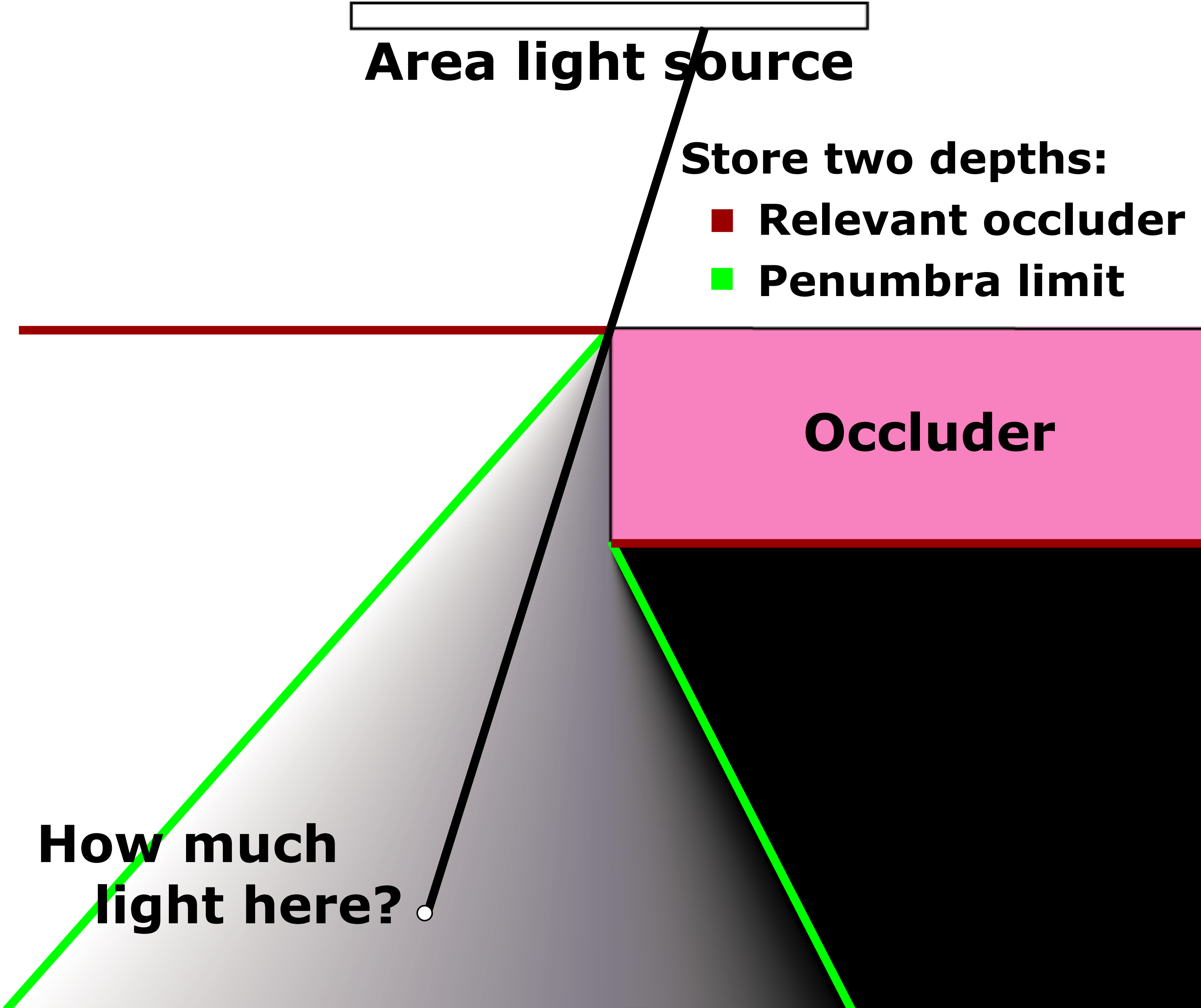
Store two depths:

- 🟥 Relevant occluder
- 🟩 Penumbra limit

Occluder

How much light here?

Penumbra Limit Map



Area light source

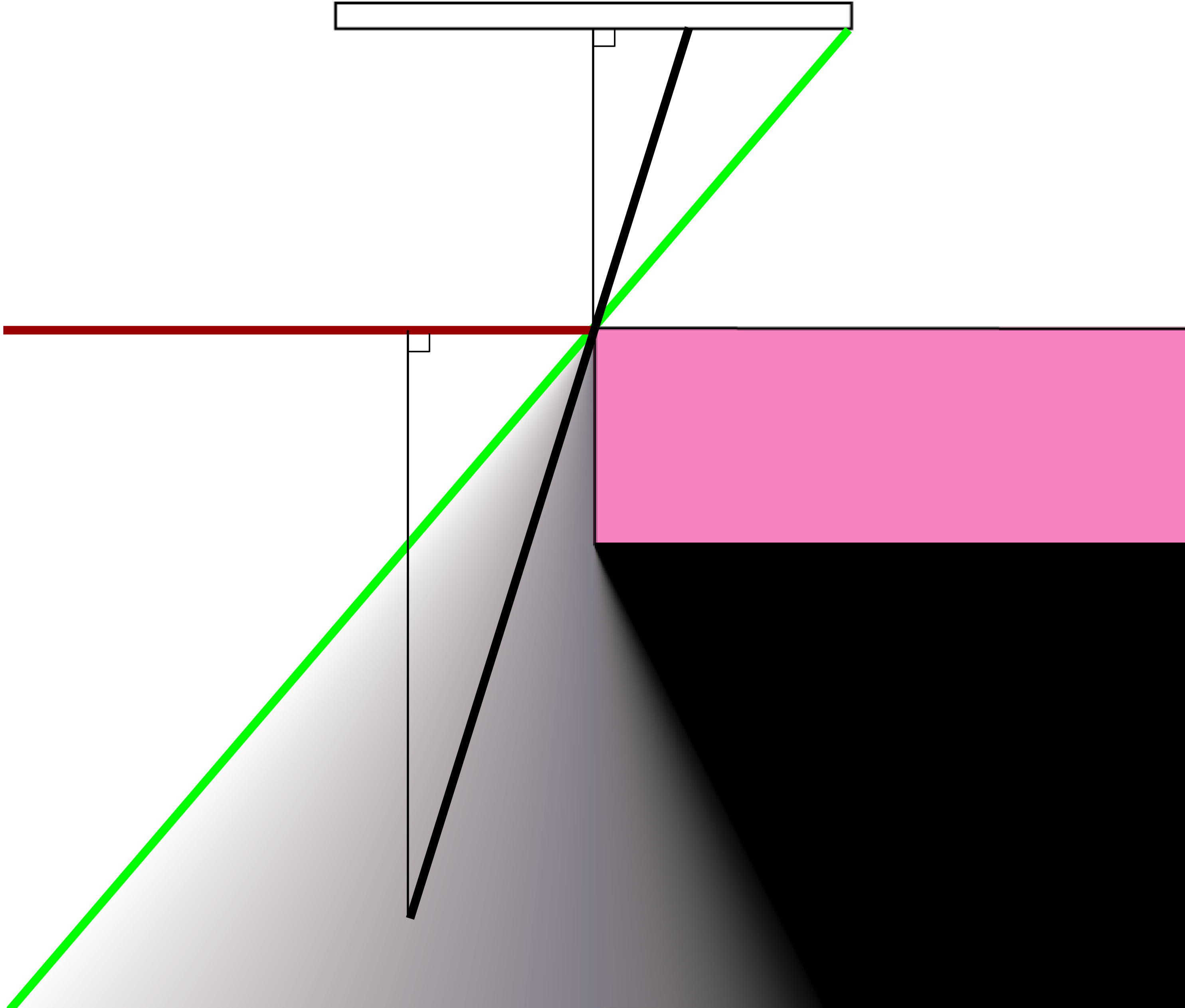
Store two depths:

- 🟥 Relevant occluder
- 🟩 Penumbra limit

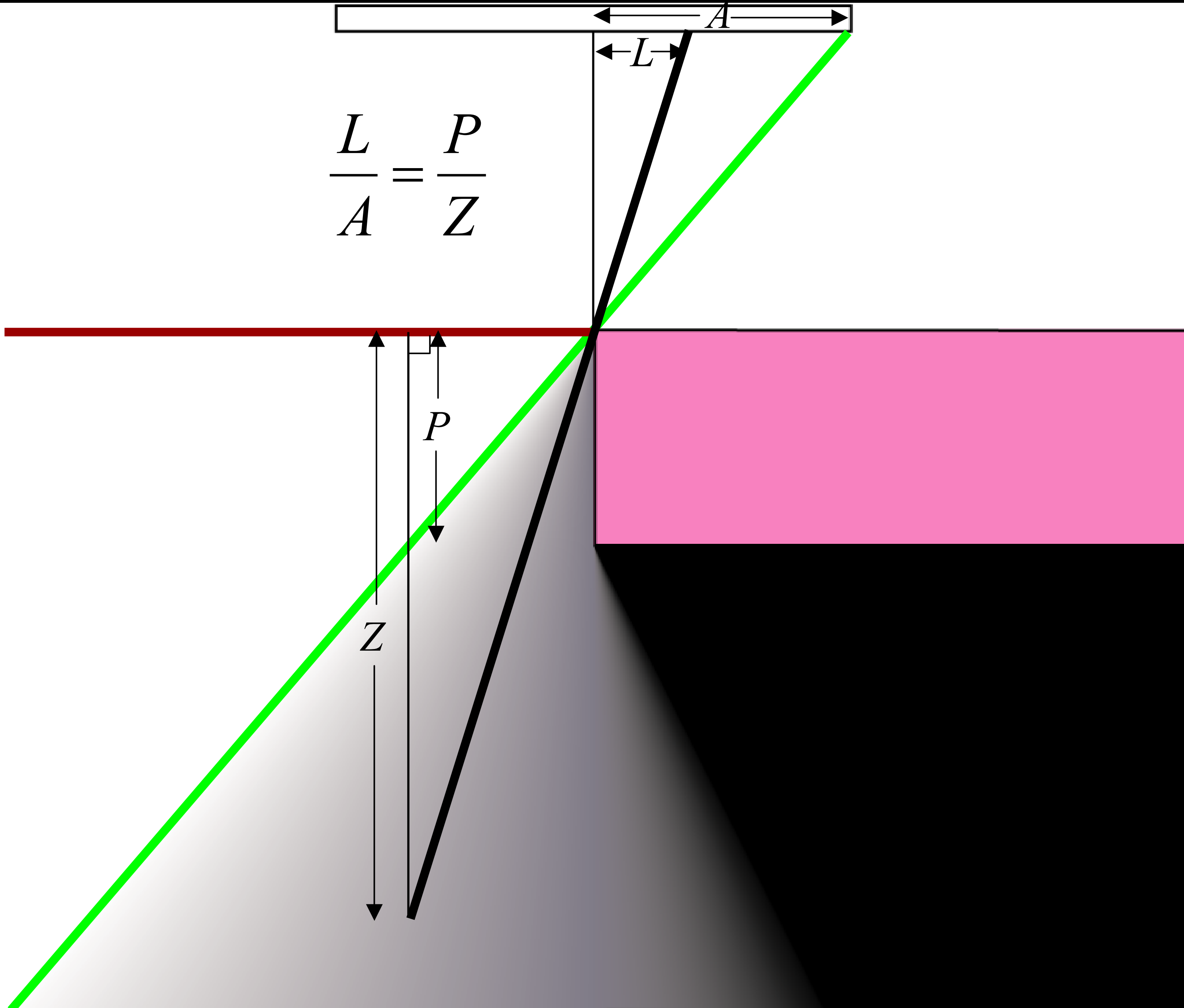
Occluder

How much light here?

Penumbra Limit Map



Penumbra Limit Map

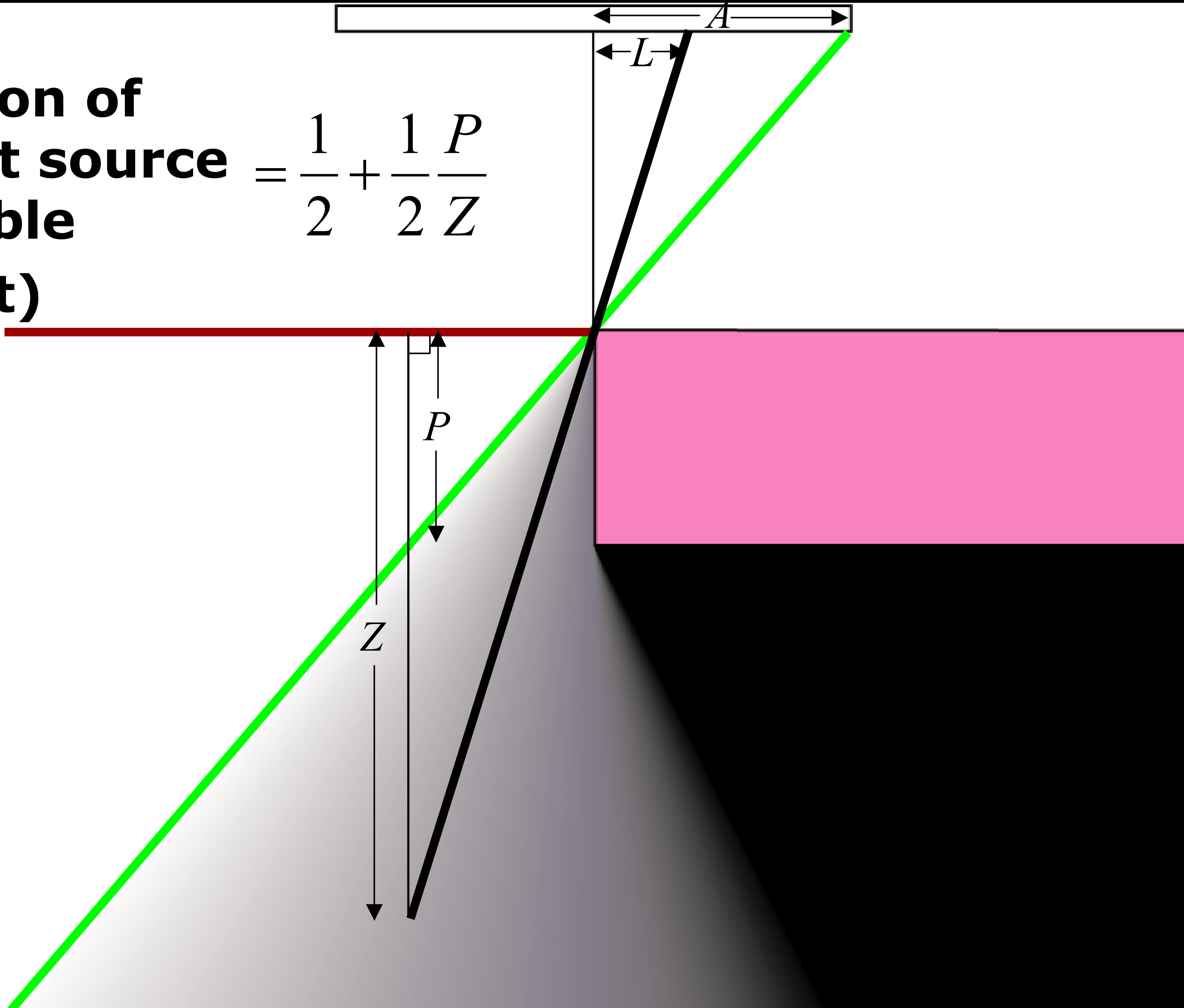


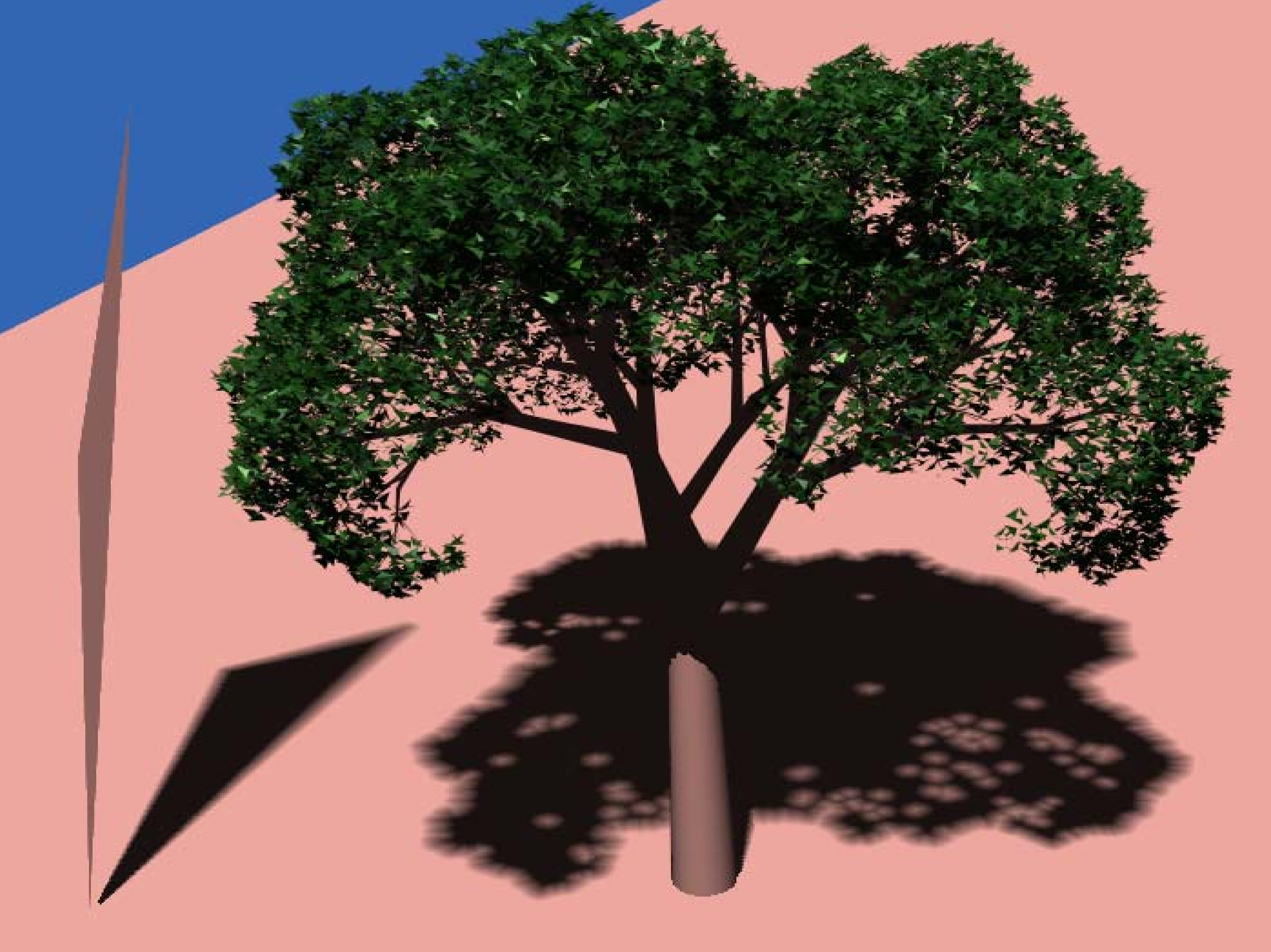
$$\frac{L}{A} = \frac{P}{Z}$$

Penumbra Limit Map

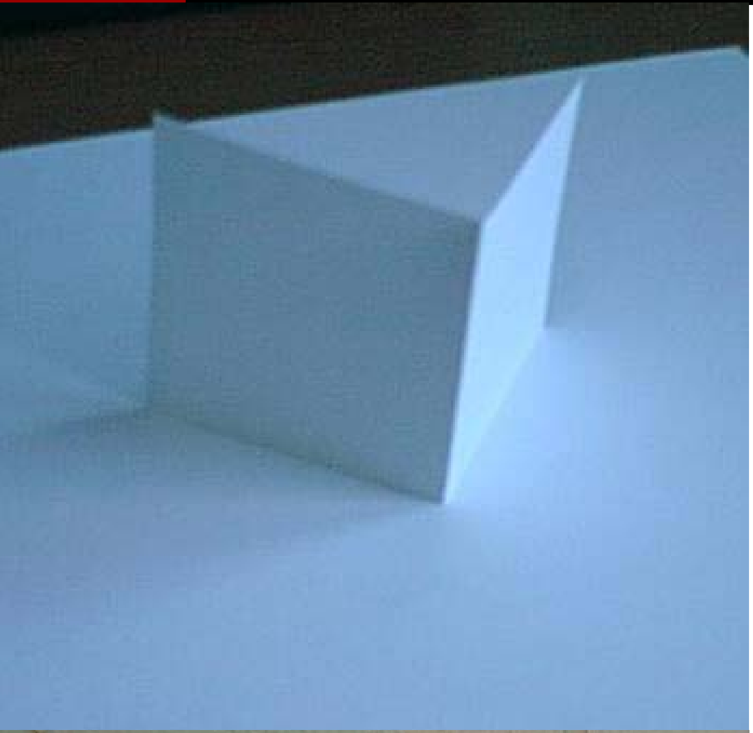
Fraction of light source visible $= \frac{1}{2} + \frac{1}{2}\frac{P}{Z}$

(exact)



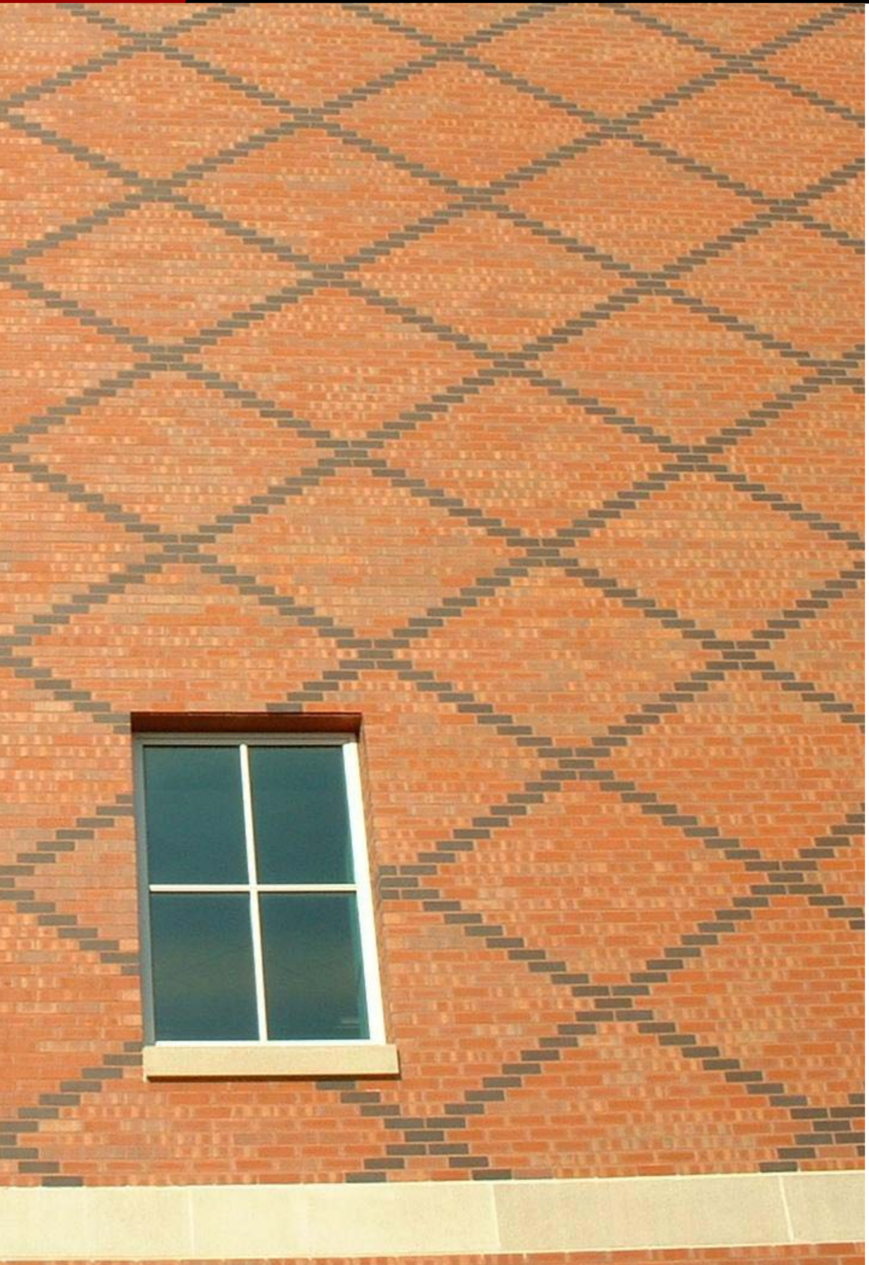


Quality: Global Illumination



- Light bounces between objects (color bleeding)
  - *Everything* is a distributed light source!
- Prior work
  - Ignore extra light
    - "Flat" look
  - Radiosity
  - Photon Mapping
  - Irradiance volume [Greger 98]
  - Spherical harmonic transfer functions

Detail: Complicated Texture



- **World's colors are complicated**
- **But can be described by simple programs**
  - **Randomness**
  - **Cellular generation**

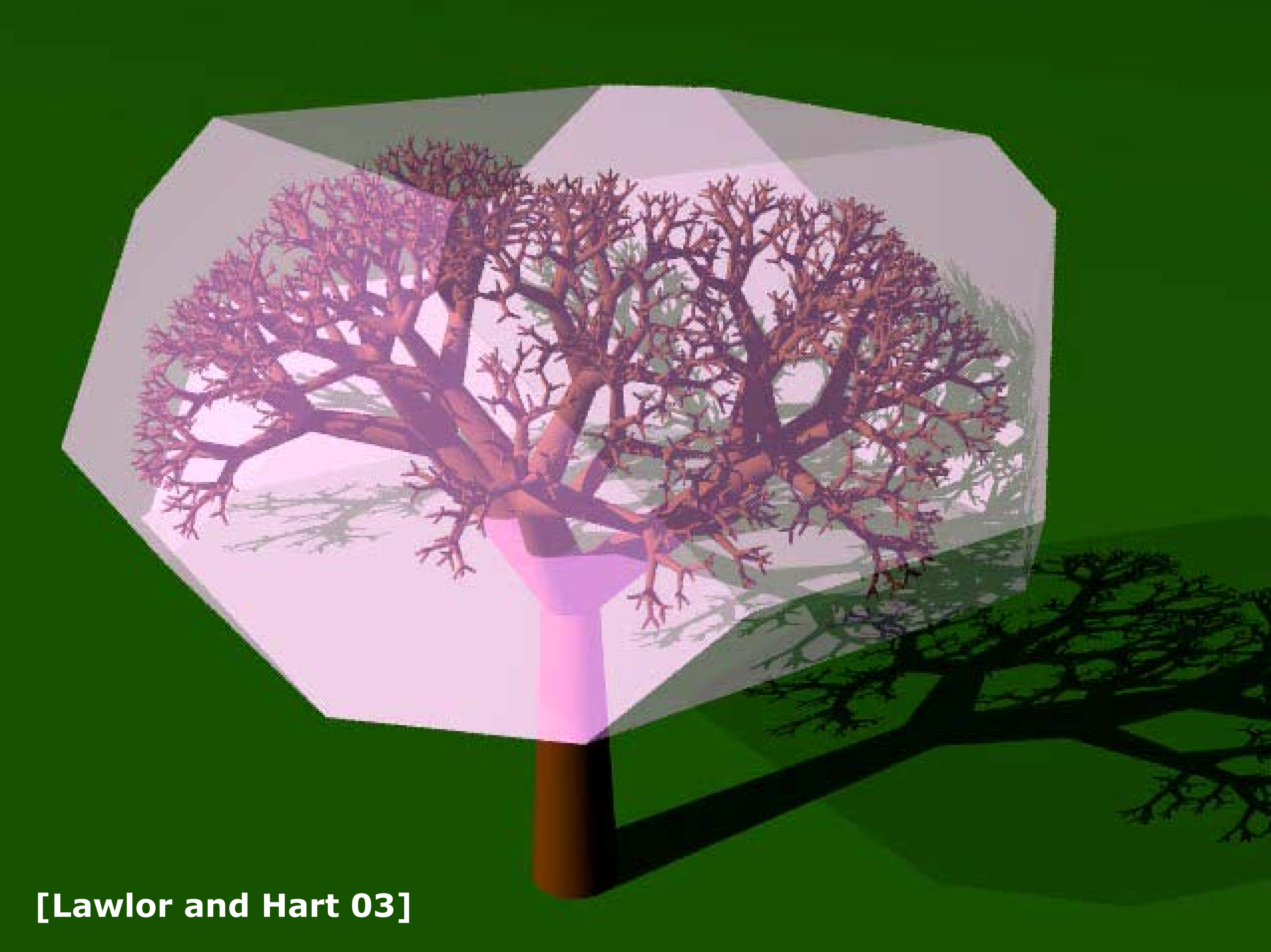
    [Legakis & Dorsey & Gortler 01]
  - **Texture state machine** **[Zelinka & Garland 02]**

- **Many are expensive to compute per-pixel, but cheap per-impostor**
  - **Multiscale noise:**
    - $O(\text{octaves})$ **for separate pixels**
    - $O(1)$ **for impostor pixels**

Detail: Complicated Geometry

- World’s shape is complicated
- But lots of repetition
- So use subroutines to capture repetition

[Prusinkiewicz, Hart]



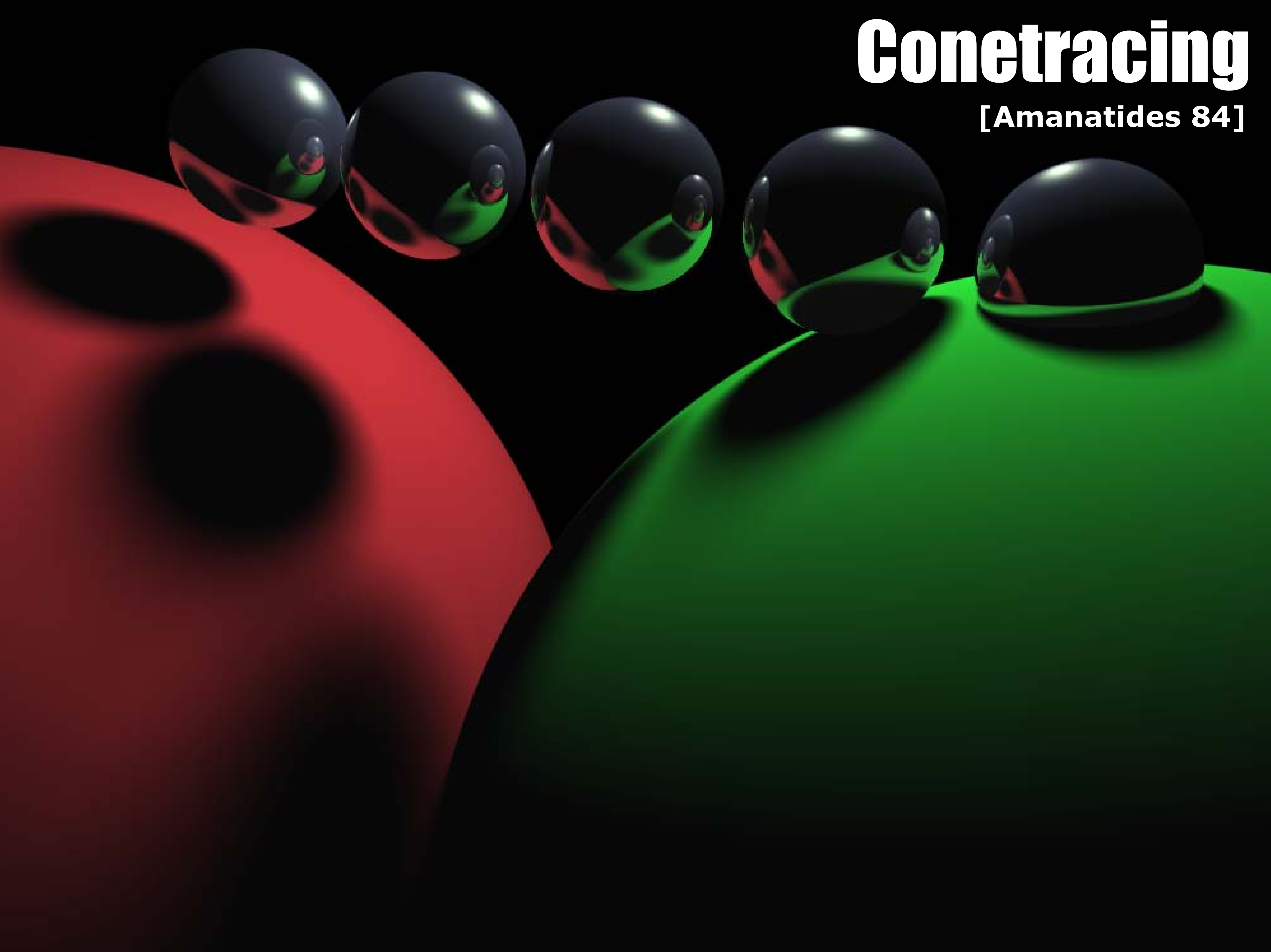
[Lawlor and Hart 03]

Scale: Kilometers

- World is really big
  - Modeling it by hand is painful!
- But databases exist
  - USGS Elevation
  - GIS Maps
  - Aerial photos
- So *extract* detail from existing sources
  - Leverage huge prior work
- Gives *reality*, which is useful

Conetracing

[Amanatides 84]



Analytical Atmosphere Model

[Musgrave 93]

Conclusions

- Parallel Impostors
  - Benefit from parallelism and benefit from impostors are *multiplied* together
- Enables quantum leap in rendering detail and accuracy
  - Detail: procedural texture and geometry, large-scale worlds
  - Accuracy: antialiasing, soft shadows, motion blur