Penumbra Deep Shadow Maps

Jean-Francois St-Amour, LIGUM – Université de Montreal
Eric Paquette, LESIA - ETS
Pierre Poulin, LIGUM – Université de Montreal

Motivation

- Shadows are a Good Thing™
- Softer is better
- Very difficult to do for complex real-time applications
- Current methods are:
  - Slow with high-quality
  - Fast with lower quality

Plan

- Previous Work
- Introduction to PDSM
- PDSM Construction
- Rendering
- Results
- Conclusion

Previous Work

- « Real-time » methods
  - Rendering precomputed soft shadows in real-time
    - Multiple Shadow Maps [Brotman-Badler 84]
    - Layered Attenuation Maps [Agrawala et al. 00]
  - Rendering dynamically computed soft shadows
    - PCF [Reeves et al. 87]
    - Smoothies [Chan-Durand 03]
    - Penumbra Maps [Wyman-Hansen 03]
    - Penumbra Wedges [Assarsson – Akenine-Moller 03]

Previous Work

- Two classes, two goals
  - Real-time dynamic soft shadows
    - Fast rendering
    - Dynamic scenes
    - Tradeoff in quality
      and ultimately max scene complexity

Previous Work

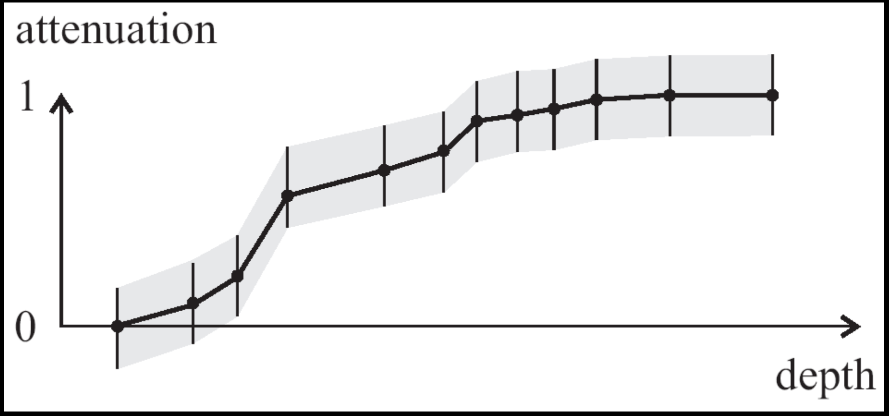
- Two classes, two goals
  - Pre-computed soft shadows
    - Real-time rendering
    - Limited to static scenes because of precomputation

Introduction to PDSM

- We propose a method to bridge the gap
  - High-quality precomputed soft shadows
    - Shadows cast by static objects
    - Real-time rendering using GPU
  - Seamless integration of dynamic objects
    - Objects inserted after shadow computation are correctly shadowed
    - Must however create their own shadows

Introduction to PDSM

- How?
  - Using Deep Shadows Maps [Lokovic-Veach 00]
    - Attenuation value for all of 3D space covered by light
    - Cumulative occlusion



  - But with penumbra information

Introduction to PDSM

- What we need
  - Construction of a DSM with penumbra information
    - Precomputation allows for a mix of software and hardware computation
  - Real-time rendering using the PDSM
    - Efficient storage
    - Rapid evaluation
    - RT requires pure hardware computation

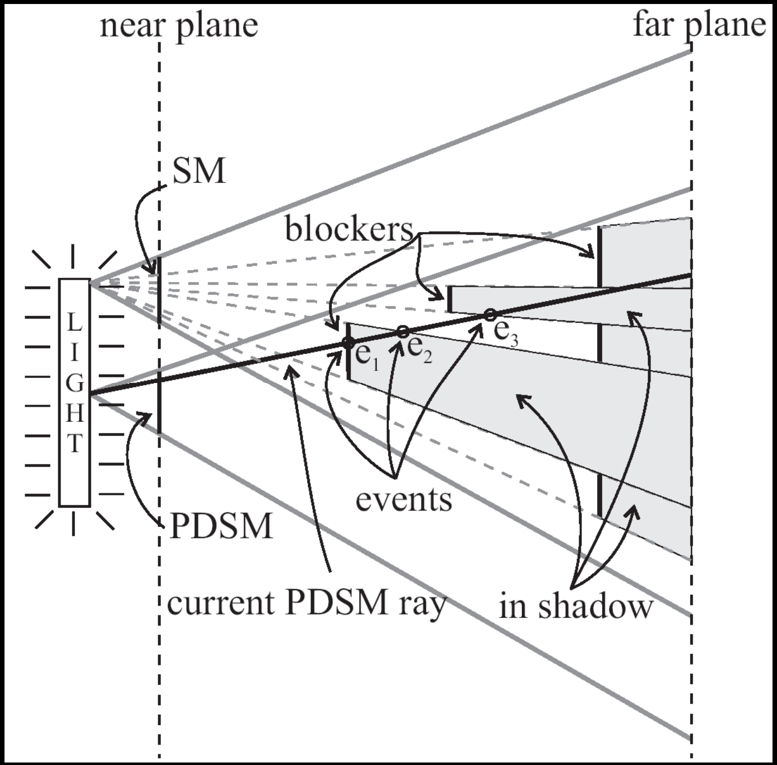
PDSM Construction

- What we want to do
  - Take multiple sample views on the light source and merge them
    - Like the LAM algo, but not really
    - Like the DSM algo, but not really

    - We want to combine their respective goals
      - Merge multiple shadow map info
      - Store attenuation function for all of light's FOV

PDSM Construction

- Overview



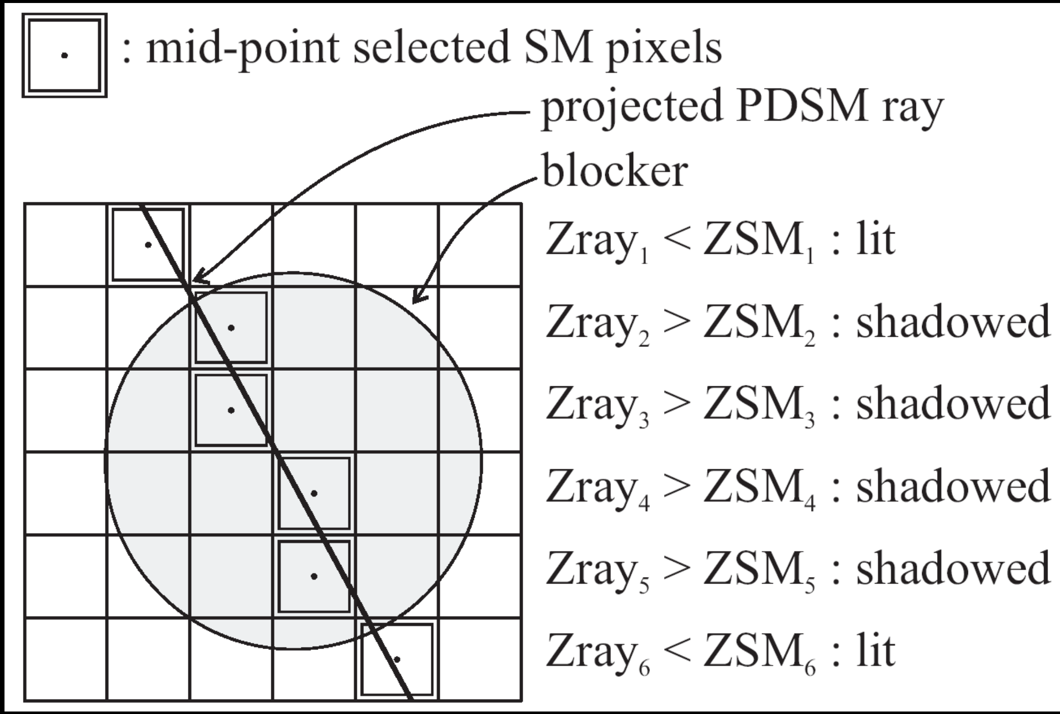
PDSM Construction

Algorithm 1: PDSM construction.

```
1 Generate k random sample points on the light source.
  foreach sample point do
2     Compute a shadow map (SM).
      // Merge the SM information to the PDSM.
      foreach PDSM pixel do
3         Compute the associated 3D PDSM ray.
4         Project this PDSM ray in the SM.
          foreach SM pixel traversed by the ray do
5             if visibility changed then
6                 Insert an event into the PDSM.
```

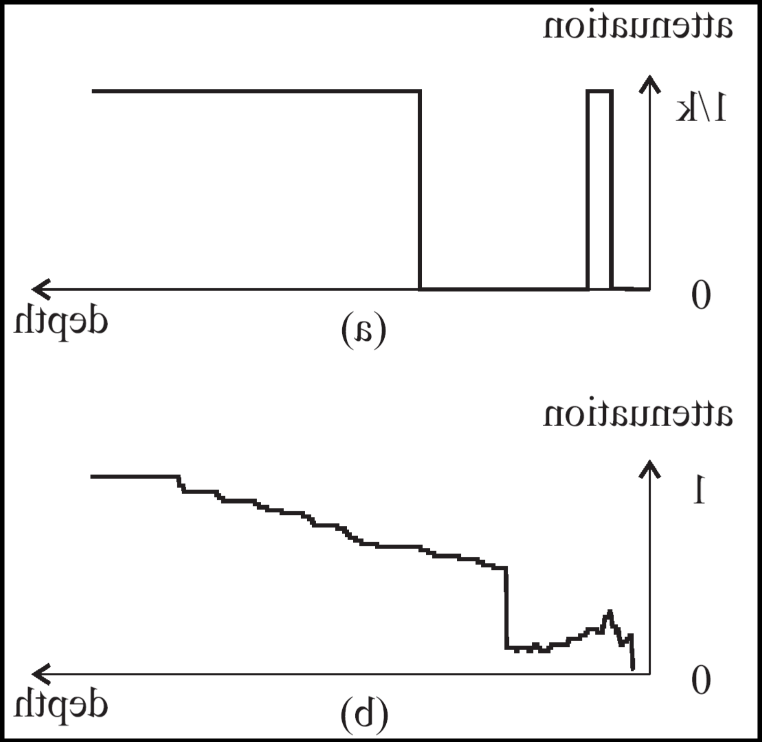
PDSM Construction

- Scan-conversion into depth buffer to find visibility events



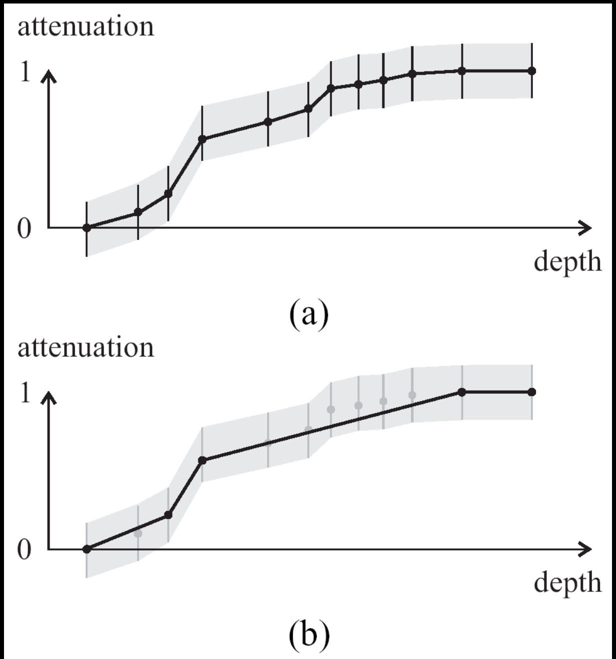
PDSM Construction

- Merging the information from one sample into the PDSM



Compression

- Guaranteed upper-bound on error



(a)

(b)

- More aggressive compression also possible

Rendering

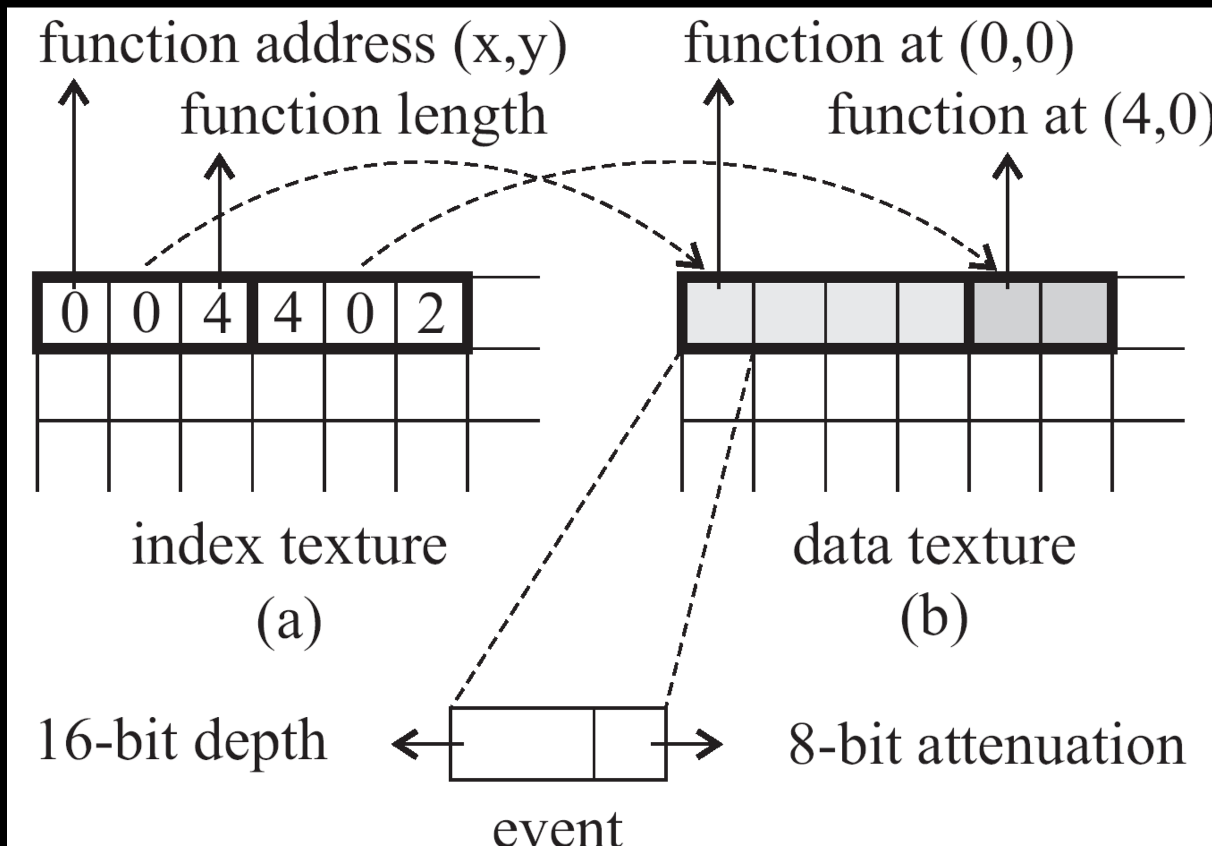
- For each point to shade, we must evaluate the PDSM function

Algorithm 2: Rendering.

foreach *3D point to shade* **do**
  // $3D\ point(x, y, z)_{world} \rightarrow (x, y, z)_{PDSM}$
  Project in the PDSM.
  // $(x, y)_{PDSM} \rightarrow f(\ )$
  Retrieve the appropriate attenuation function.
  // $f((z)_{PDSM}) \rightarrow attenuation$
  Retrieve the attenuation value.
  // $attenuation \rightarrow pixel\ color$
  Modulate the shading by this attenuation.

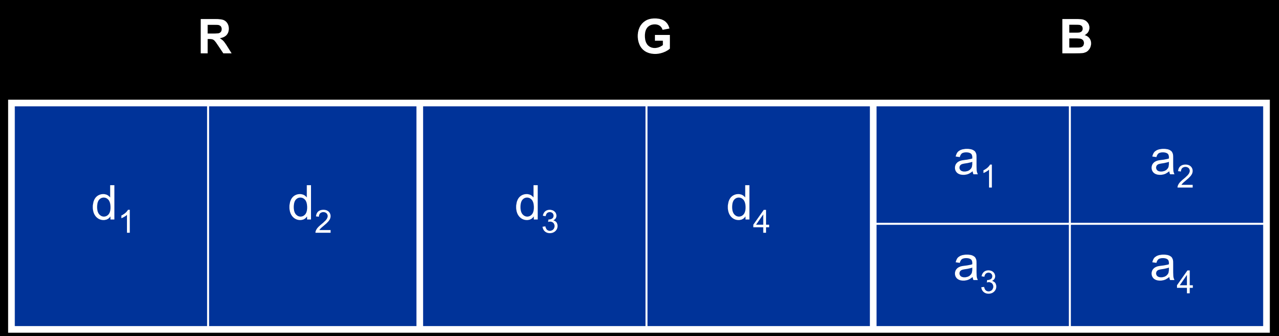
GPU Storage

- Two textures: *Index* and *Data* texture



GPU Storage

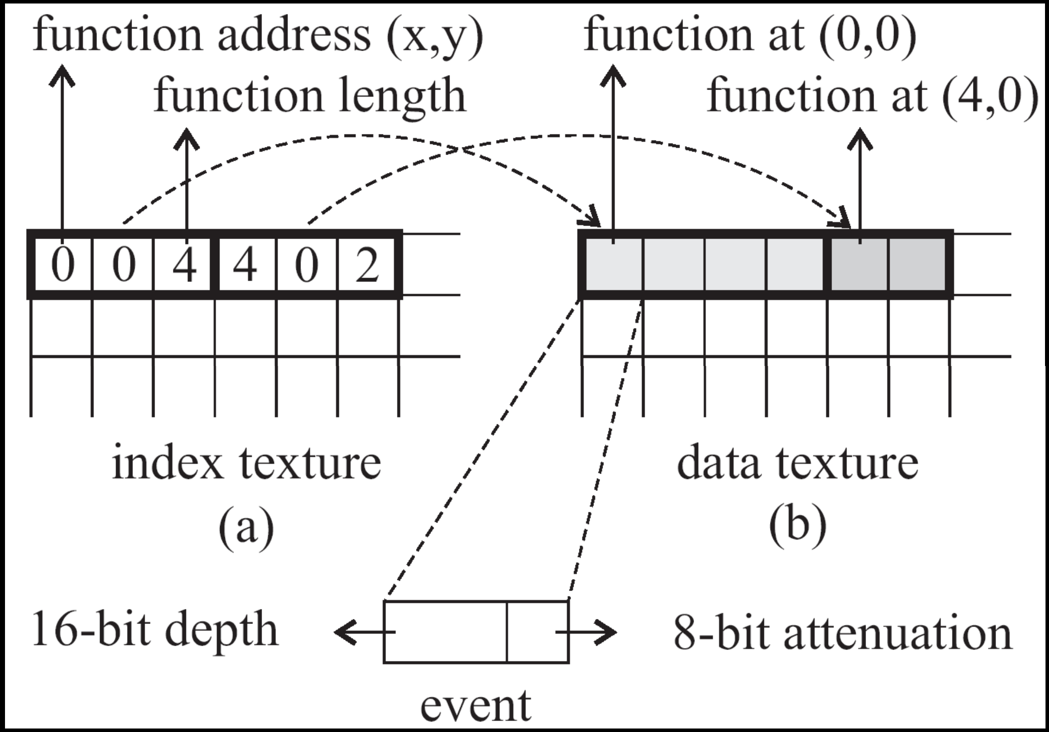
- Packing the Data texture
  - One RGB32F texel contains 4 function points



d_i : 16-bit depth a_i: 8-bit attenuation

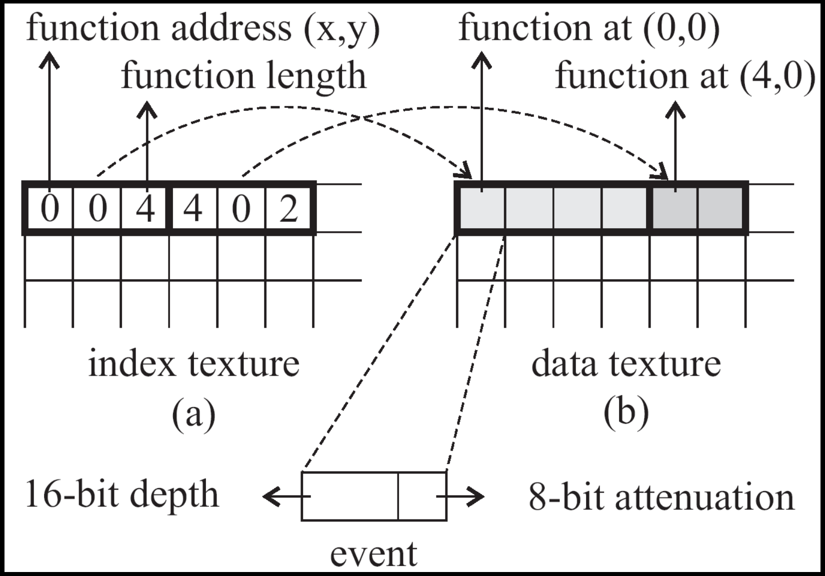
GPU Evaluation

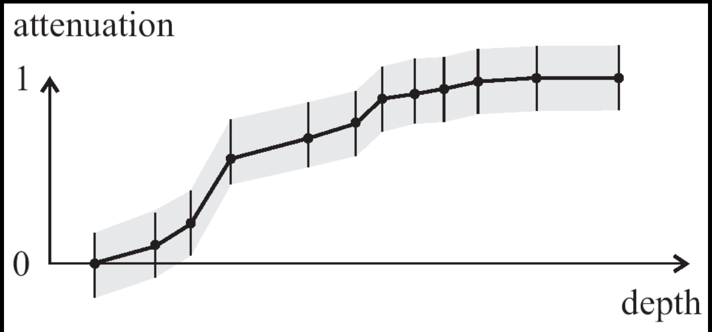
- Find the right PDSM function in the *Index texture*
  - Using regular Projective texturing



GPU Evaluation

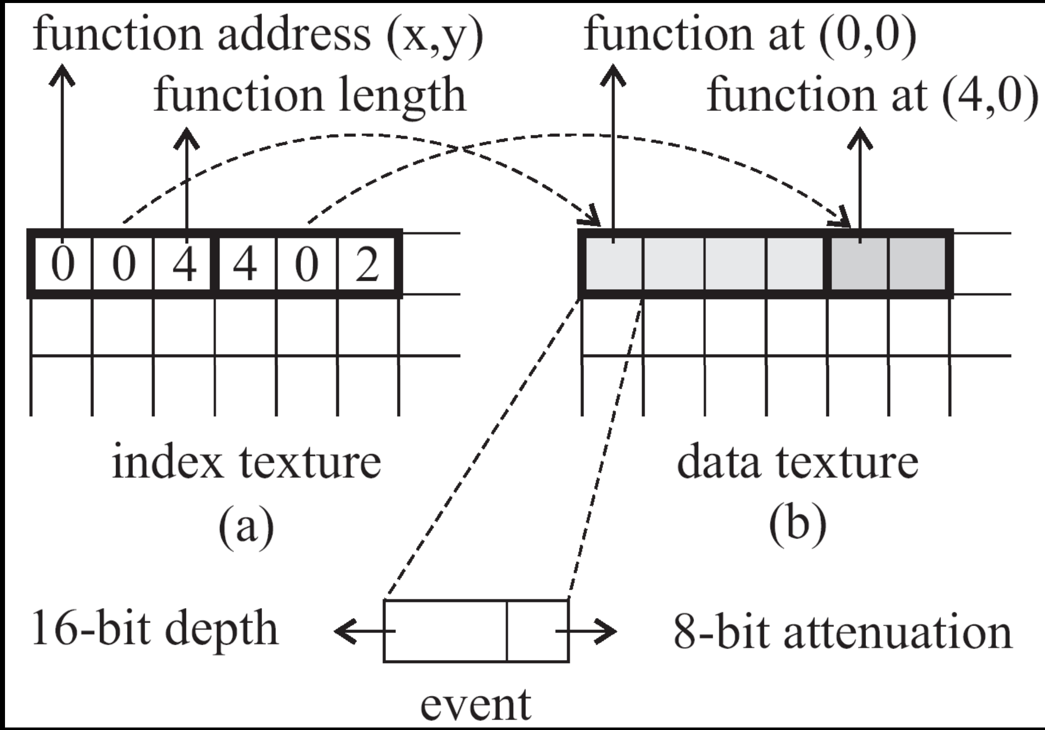
- Get the function points from the *Data texture*
  - Incremental dependant texture lookups





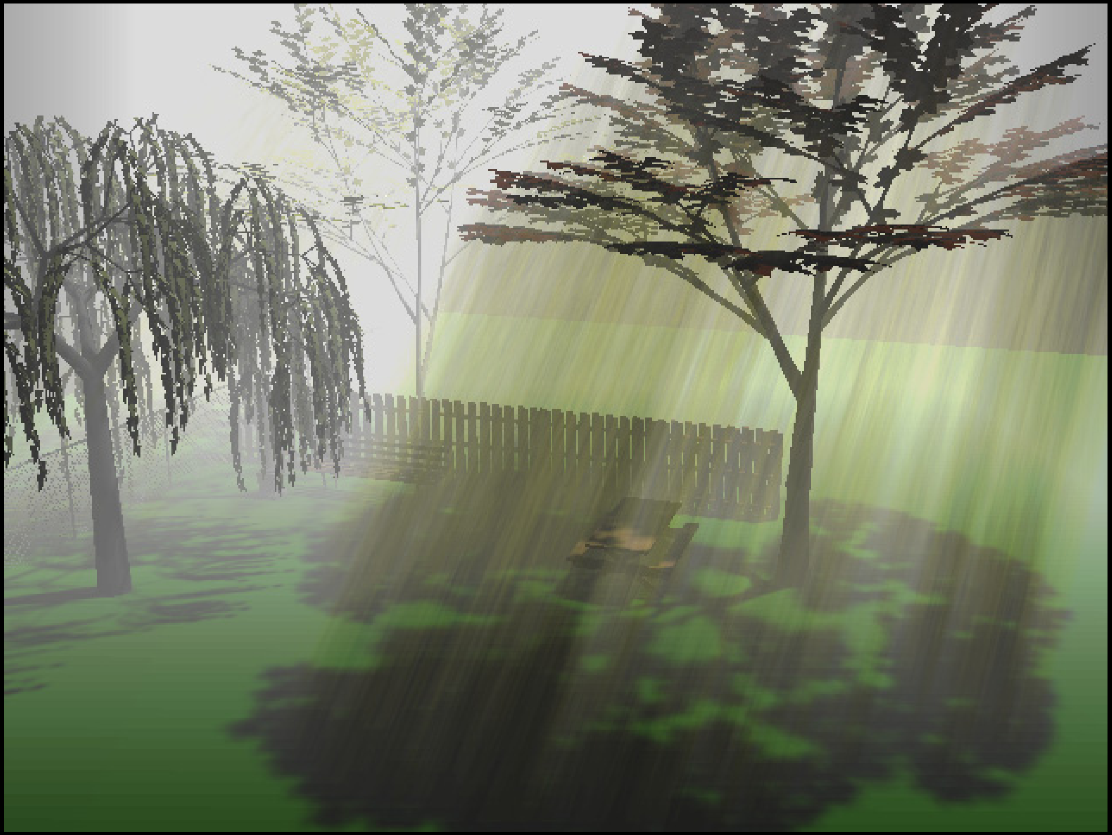
GPU Evaluation

- Advanced features require real dynamic branching at fragment level
  - Early-out during evaluation
  - Arbitrary function lengths



Results

- Video



Conclusion

- Recap
  - High-quality soft shadows for static objects
  - Dynamic object insertion
  - Real-time rendering using the GPU
    - Efficient storage
    - Rapid evaluation using the fragment processor

Conclusion

- Future Work
  - Faster construction
    - “Chunks” of PDSM rays
    - Peeling approach
  - Perceptual approach to compression
  - Enhanced light sampling function
  - PDSM approximation with very few samples

- Special thanks
  - Luc Leblanc
  - Philippe Beaudoin
  - Andrew Woo
  - NSERC

- Questions?