Parallel-Split Shadow Maps for Large-scale Virtual Environments

Hanqiu Sun
VR, Visualization and Imaging Research Centre
Dept. of Computer Science & Engineering
The Chinese University of Hong Kong

Motivation of Shadow Rendering

- Important for increasing scene realism
- Current graphics programming APIs didn’t implement shadow effects
  - Use local illumination models which don’t consider the spatial relation between objects.
- Global illumination models
  - Highly realistic rendering effect (ray tracing, radiosity, photon mapping)
  - Hard to apply for real-time applications

Shadow Rendering Techniques

- **Shadow Volume**
  - Object-space algorithm
  - Accurate
  - Computational expensive, good tessellation
- **Shadow Mapping**
  - Image-space algorithm
  - Fast rendering speed, only one extra rendering pass needed
  - Aliasing errors from insufficient texture resolution

Introduction

- Object-Based Algorithm: Shadow Volumes
  - Pros: with the screen-space resolution.
  - Cons: computationally intensive.
Introduction

Image-based Algorithm: Shadow Mapping

- Pros: widely used in real-time applications by its efficiency and generality.
- Cons: suffers from inherent aliasing problem caused by insufficient texture resolution.

What does image-based mean?

- No geometry information needed
- Suffers from aliasing artifacts
- Multi-pass rendering
  - Render the scene two passes rather than single one
  - Render the scene from the light’s point-of-view
    - Extract depth buffer as a texture, i.e. shadow map
    - Render the scene normally with the depth map

Shadow Mapping

Pass 1: depth testing in light’s point-of-view

- Current depth buffer is a “depth map” or “shadow map”
- Essentially a 2D function indicating the distance of the closet pixels to the light

Pass 2: render shadows with shadow map

- Transform each pixel to light’s space
  - \((x, y, z)\) in viewer’s space
  - \((x', y', z')\) in light’s space
- Compare \(z'\) to the depth value at \((x', y')\) stored in shadow map

- “\(\gamma\)” shadowed, “\(\gamma'\)” illuminated
Aliasing Errors Analysis

Where do aliasing errors come from?

- light perspective aliasing
- projective aliasing

Aliasing in Shadow Mapping

Anti-aliasing is required in shadow mapping due to the image-based nature.

Perspective Aliasing

- View dependent, under-sampling for near areas, super-sampling for distant areas.
- The only kind of aliasing can be alleviated by a global transformation like a perspective warping.

Projection Aliasing

- Reducing projection aliasing force us to abandon hardware-acceleration. (pure software solution)
- In this presentation, we only address on the techniques reducing perspective aliasing errors.
Main Research Directions

- **Geometry approximation**
  - Focus shadow map on visible objects
  - Approximate the bounding shape of the intersection between light frustum and view frustum

- **Perspective shadow maps**
  - Render shadow map in post-perspective space rather traditional view space

- **Hierarchical data structure**
  - Adaptive resolution to viewer’s position
  - Inconveniently used on current hardware

- **Hybrid shadow maps with shadow volume**
  - Take both merits and drawbacks

Recent: EUROGRAPHIC’04

Problems for Current Methods

- Re-parameterize shadow map for the whole scene
  - Treat all objects in the same manner

- Single shadow map with huge resolution
  - Impractical to memory-sensitive applications

- Geometry approximation approaches still need improvement
  - More accurate approximation, more resolution increasing

Parallel-Split Shadow Maps

- Use multiple smaller shadow maps rather single huge one
  - Total memory requirement is less, but with better shadow qualities

- Use multiple planes parallel to view plane to split the scene into different depth layers
  - Re-parameterization is proceeded in each layer rather the whole scene
Parallel-Split Shadow Maps

Advantages
- Suitable for large-scale environments
- Make the distribution of aliasing errors along depth range more even
- Less texture memory requirement
- Higher resolution utilization ratio
- More compact bounding shape
- Extendable framework easily integrated with other shadow mapping algorithms
- Any algorithm can be applied to any layer

Implementation Steps
- Step 1: view frustum split
  - How to determine the split planes’ positions?
- Step 2: light’s view plane split
  - Calculate the shadow map viewport for each split part, i.e. shadow map window
Implementation Steps

Step 3: focus shadow map windows
- Increase resolution for each shadow map

Step 4: PSSMs rendering
- Resolutions configuration

Step 5: shadowed scene rendering
- Need multiple rendering passes for shadow maps generation
  - Multiple Rendering Targets (OpenGL 2.0/DirectX 9)
- Need multiple rendering passes for final scene-shadows rendering
  - Pixel Shader
- Practical number of split parts
  - Split-1, split-2

PSSMs on DX9 Hardware

DirectX-9 Level GPU acceleration (e.g. GeForce 6800)

For each pixel $p(x, y, z)$ in pixel shader, if

the associated shadow map $T_i$ and texture coordinates $tex_i$ should be selected to do shadow determination.

Partially GPU-accelerated: by using pixel shader on DX9 GPU, scene-shadows rendering only needs single one rendering pass.
PSSMs on DX10 Hardware

DirectX-10 Level GPU acceleration (e.g. GeForce 8800)

- **render target array** allows rendering the object into multiple render targets and depth stencil textures simultaneously.
- **geometry shader** controls which render target every primitive will be sent to and applies the appropriate transformation matrix to the vertex coordinates.

Fully GPU-accelerated: with the above two brand new features, both generating shadow maps and scene-shadows rendering only require single one rendering pass!

Experimental Results

![Experimental Results Image]

Parallel-Split Shadow Maps

- **Uniform Split Scheme**
  - the aliasing distribution is same as that of standard shadow maps.
- **Logarithmic Split Scheme**
  - this scheme produces the theoretically even distribution of perspective aliasing errors.
- **Practical Split Scheme**
  - this scheme produces the "moderate" sampling densities over the whole depth range.

View Frustum Split
Uniform Split Scheme

\[ \{ C_i^{\text{uniform}} \} \quad 0 \leq i \leq m \]
\[ C_i^{\text{uniform}} = n + (f - n) \frac{i}{m} \]
- the sampling densities are same as standard shadow maps.
- under-sampling for the objects near the viewer, over-sampling for the distant objects.
- worst case.

Logarithmic Split Scheme

\[ \{ C_i^{\text{log}} \} \quad 0 \leq i \leq m \]
\[ \frac{dz}{zds} = \rho \Rightarrow s = \int_0^1 ds = \int_0^1 \frac{1}{\rho} f_z dz = \frac{1}{\rho} \ln \left( \frac{z}{n} \right) \]
\[ s_i = s(C_i^{\text{log}}) = \ln \left( C_i^{\text{log}} / n \right) / \ln (f / n) \]
- since this scheme produces the theoretically even aliasing distribution.
- in practice, too much resolution assigned to the objects near the viewer.

Practical Split Scheme

\[ \{ C_i \} \quad 0 \leq i \leq m \]
\[ C_i = (C_i^{\text{uniform}} + C_i^{\text{log}}) / 2 \]
- moderate the optimal and worst sampling densities.
- no need for the complicated analysis.
Performance

- Real-time for PSSM(2), PSSM(3), PSSM(4).

Pixel Shader Program

```c
float4 PixelShader_Program(PS_INPUT IN): COLOR
{
    // other statements
    float bias = 0.004;
    bool stop = false;
    float shadow_value = 0;
    for (int i = 0; i < 4 && !stop; i++)
    {
        if (pos.z / pos.w <= splitPositions[i])
        {
            float depth = IN.tex[i].x / IN.tex[i].w;
            float depth_SM = tex2Dproj(PSSM_Sampler[i], IN.tex[i]);
            shadow_value = (depth < depth_SM + bias);
            stop = true;
        }
    }
    return shadow_value * color;
}
```
PSSMs Applications

Adopted in commercial games! PSSM[4; 1Kx1K]

Visual Results

Screenshots from the game Dawnspire: Prelude (www.dawnspire.com) courtesy of Silent Grove Studios

PSSMs Applications

ожет

The famous 3D graphics OpenGL-based SDK OpenSceneGraph

http://www.openscenegraph.org/ start to support PSSMs.

The game engine Hammer Engine

http://www.hmrengine.com/en/ integrates our PSSMs scheme

Several implementations (including OpenGL- and DirectX-based) of PSSMs have been developed by volunteers.

Direction-dependent Parameterization

Problems of existing parameterizations:

- The “optimal” distribution of aliasing errors is achieved only in the ideal case, i.e. the light direction and view direction are orthogonal.
- No generalized theoretical framework to analyze the aliasing distribution over the whole depth range in general cases.
Perspective Warping

- Generate shadow maps in the post-perspective space.
- Increase sampling densities for near objects.

Freedoms of Warping Frustum

- The essential difference among variant perspective parameterizations is the selection of near plane.
- Warping direction.

Near Plane Selection

- Near plane of the warping frustum determines how strong the warping effect is.
- Notice that stand shadow map also can be thought of as a special perspective parameterization! Where the near plane is set at infinity.
- Too small near plane will cause an over-strong warping effect. In practice, enlarge near as possible.
Warping Direction

- It's important! The type of lighting source will be frequently changed after perspective warping!
- The frequently changed light types also result in the mapping singularities.

A smart selection of the warping direction was proposed in [Wimmer, M. et al. 2004]. The warping direction is set to parallel to the shadow map plane.
- All types of lighting sources are converted to directional ones. No mapping singularities produced!

Problems (1)

- Optimal distribution of aliasing errors is achieved in ideal case only
  - previous work $\lambda = \text{constant}$ (or not direction-optimal)

Our Goals

- Re-explain existing persp. reparams.
- Develop algorithms that stabilize optimal error distribution for given constraints
FTSM

Trapezoidal Shadow Maps (TSMs)
- use a trapezoid to approximate the view frustum as seen from light.
- map the first 50% depth range to 80% fraction of shadow map.

Problem 1 of TSM
- The focus region is not preserved in general case.

Problem 2 of TSM
- unexpected stretching on the foreground

TSM vs. FTSM
- Aliasing distributions
  - Both x-direction and z-direction are considered
TSM vs. FTSM

Direction-dependent Parameterization

Generalized Linear Parameterization

- Linear aliasing distribution.
  - $F = k (k$ is constant)
  - Linear distribution requires $\varphi(\lambda, \theta) = 0$

$$\lambda = \begin{cases} 
\lambda_1 (\theta) = \frac{n - \frac{a}{\tan \theta}}{f - n + \frac{a}{\tan \theta}}, & |\theta| \in \left[\frac{f_{\text{ov}}}{2}, \frac{\pi}{2}\right] \\
\lambda_2 (\theta) = \frac{f \tan \theta - b}{2b}, & |\theta| \in \left[0, \frac{f_{\text{ov}}}{2}\right]
\end{cases}$$

- Our selection depends on rather a constant used in prior work.

- $\lambda_1$ and $\lambda_2 \leq 0$
  - The near plane should be positive!
  - The linear condition can NOT be satisfied in all cases!
  - The field-of-view is usually not very wide, so we can say that the linear parameterization can be achieved in MOST general cases.
How to modify the equation for coding?
- We need to determine a constant $\lambda$ and a new function $\lambda_2(\theta)$ when $0 \leq \theta \leq \gamma$. 

Why does $\gamma$ appear?
- Notice that $\lambda_2(\theta)$, which will cause all pixels mapped into single texel! The warping effect is too strong, no any sense in practice.

$$
\lambda_{GLPR} = \begin{cases} 
\lambda_1(\theta) = \frac{n - \frac{a}{\tan(\theta)}}{f - n + \frac{a + b}{\tan(\theta)}}, & |\theta| \in [\gamma, \frac{\pi}{2}] \\
\lambda_2^*(\theta), & |\theta| \in [0, \gamma]
\end{cases}
$$

How to select the function $\lambda_2(\theta)$?
- Continuous $\lambda$ transition at $\gamma$.
  - To keep the consistent transition of shadow qualities, continuous $\lambda$ transition at $\theta = \gamma$ should be guaranteed.
  - GLPR converges to SSM as $\theta$ goes to 0.
  - All perspective parameterizations degrade to standard shadow maps as the light direction goes to parallel to the view direction.

In our system, the following function that satisfies the previous two criteria.

$$
\lambda_2^*(\theta) = \lambda_1(\gamma)/\sin(\frac{\theta}{2\gamma}) = \frac{n - \frac{a}{\tan(\gamma)}}{f - n + \frac{a + b}{\tan(\gamma)} \sin(\frac{\theta}{2\gamma})}
$$

The selection $\lambda$ for GLPR

$$
\lambda_{GLPR} = \begin{cases} 
\lambda_1(\theta) = \frac{n - \frac{a}{\tan(\theta)}}{f - n + \frac{a + b}{\tan(\theta)}}, & |\theta| \in [\gamma, \frac{\pi}{2}] \\
\lambda_2^*(\theta) = \lambda_1(\gamma)/\sin(\frac{\theta}{2\gamma})), & |\theta| \in [0, \gamma]
\end{cases}
$$

How to determine $\gamma$?
- Make sure $\lambda(\theta)$ is not very small.
  - why? An over strong warping will be caused by a too small near plane selection. In practice, we’d better let $\lambda$ be larger as possible.
- In our current implementation, we select a $\gamma$ satisfying
  $$
  \lambda(\gamma) \geq 0.7 - \frac{n}{f - n}
  $$
Virtual Chess-board Scene

Conclusion

- Shadow rendering techniques
  - Shadow volume and Shadow mapping
- Shadow mapping aliasing errors
- Problems for current shadow mapping algorithms
  - Treat all objects in the same manner
  - Huge memory requirement in some cases
  - Need more precise bounding shape

Conclusion

- Parallel-Split Shadow Maps
  - Regard the scene as depth layers
  - Render shadow maps for split parts
- Advantages
  - Suitable for large-scale virtual environments
  - More compact bounding shape
  - Less texture memory requirement
  - A “layered” framework easily integrated with other shadow mapping algorithms
Conclusion

- We have formulized the Generalized Linear Perspective Reparameterization GLPR in shadow mapping

Future work

- Analysis for distribution of aliasing errors along arbitrary warping direction and light sources
- Investigate effective integration with other shadow mapping algorithms