Algorithmen für die Echtzeitgrafik

Algorithmen für die Echtzeitgrafik

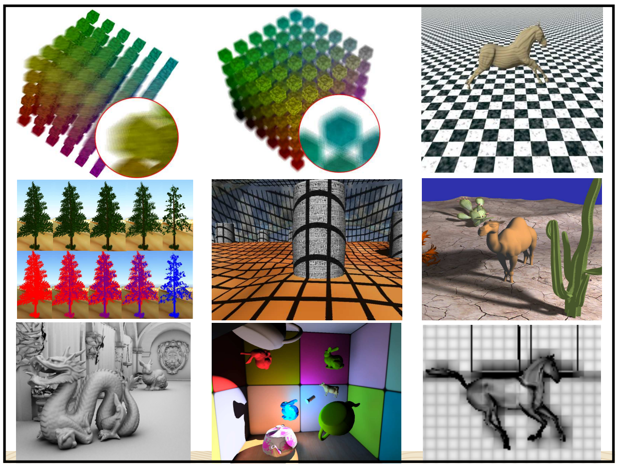
Daniel Scherzer
scherzer@cg.tuwien.ac.at

LBI Virtual Archeology

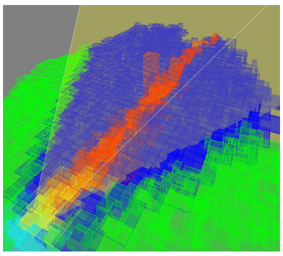
Temporal Coherence

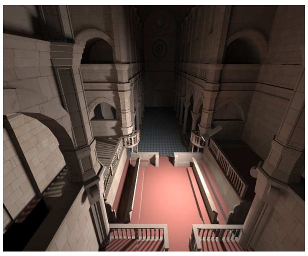
Syllabus

1. Introduction
2. Image space
   1. Theory: Image-space reverse reprojection
   2. Applications
3. Object space



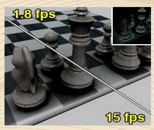
Object Space

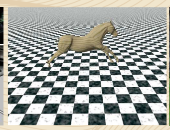


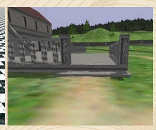


Temporal Coherence

Introduction

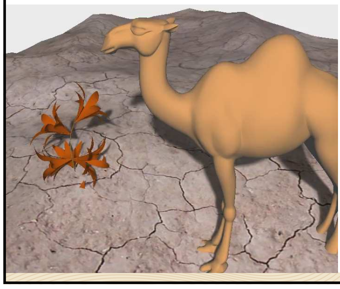


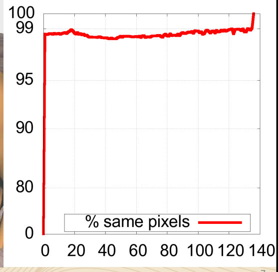




What is Temporal Coherence

- Information that stays valid for multiple queries
- Min 60 FPS in RTR → high temporal coherence





Objectives of Using Temporal Coherence

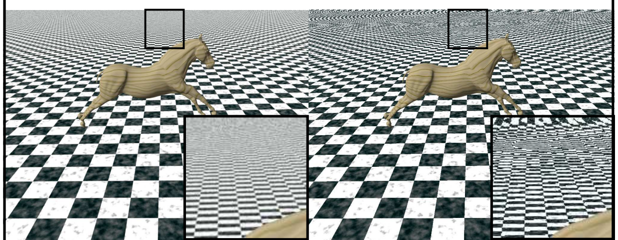
- Speed up
- Increase in quality
- Reducing temporal aliasing

Objectives of Using Temporal Coherence

- Speed up: distribute workload over several frames

Objectives of Using Temporal Coherence

- Increase in quality
  - Incorporate calculations from previous frames



Objectives of Using Temporal Coherence

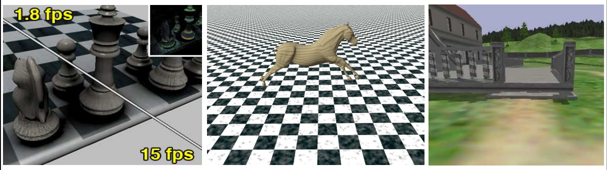
- Reducing temporal aliasing (flickering)
  - Avoid sudden changes in coherent regions

Objectives of Using Temporal Coherence

Conclusion

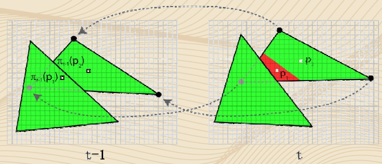
- Idea of temporal coherence (TC)
- Next:
  - Image-Space Real-Time Reverse Reprojection

speed **quality** **stability**



Temporal Coherence

Image-Space Real-Time Reverse Reprojection



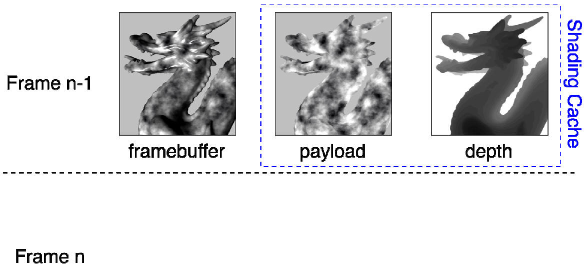
Outline

- Image-space spatio-temporal data structure
- Reverse reprojection cache
- Implementation
- Determining what to reuse
- Analysis

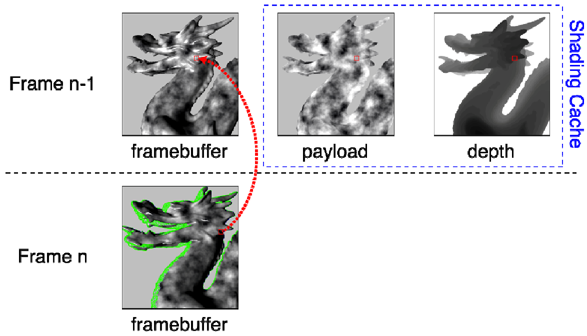
Outline

- Image-space spatio-temporal data structure
- Reverse reprojection cache
- Implementation
- Determining what to reuse
- Analysis

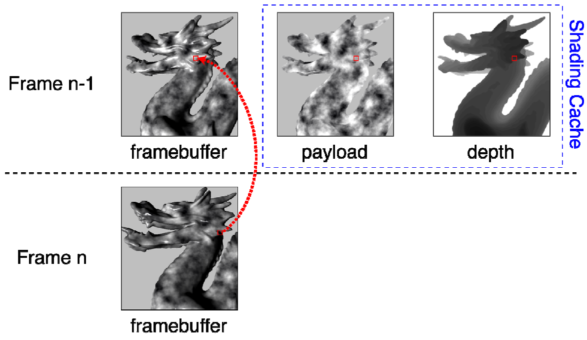
Image space shading cache



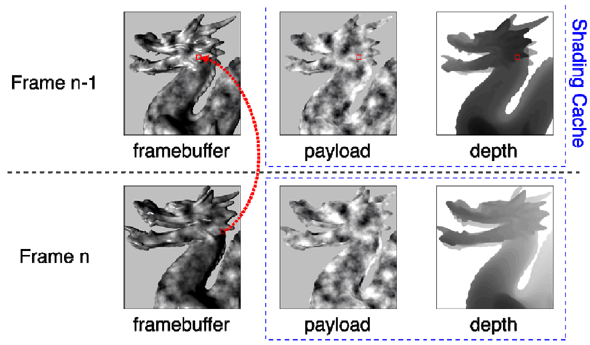
Reprojection



Reprojection



Reprojection

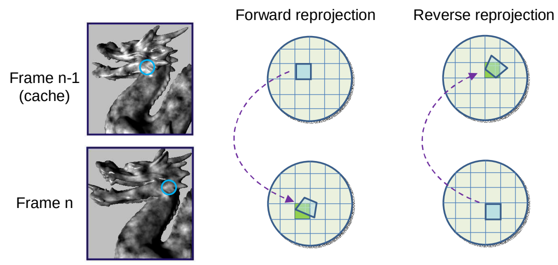


Outline

- Image-space spatio-temporal data structure
- Reverse reprojection cache
- Implementation
- Determining what to reuse
- Analysis

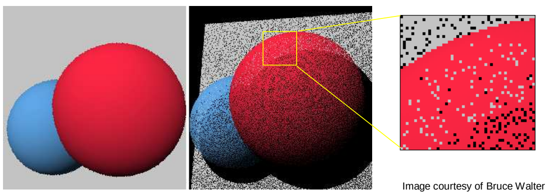
Reprojection

- No exact 1-to-1 pixel mapping (bijection) exists



Forward reprojection

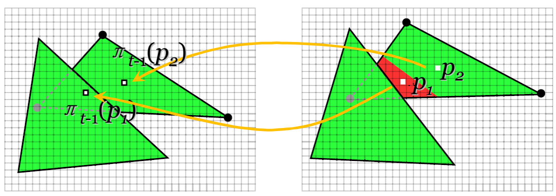
- Requires forward motion vectors
- Holes and gaps need filtering with depth culling
- Difficult to implement with DX9/10 level hardware



cache ($f_{t\text{-}1}$) new frame (f_t)

Reverse reprojection [Nehab 06/07, Scherzer 07]

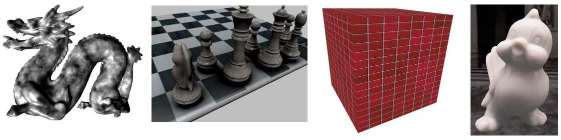
- Reprojection operator $(x', y', z') = \pi_{t\text{-}1}(p)$
- Cache: $f_{t\text{-}1}$, cache depth: $d_{t\text{-}1}$
- Test if $z' \approx d_{t\text{-}1}(x', y')$ for occlusion



cache ($f_{t\text{-}1}$) new frame (f_t)

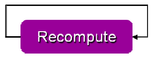
Case study: Pixel shader acceleration

- Today: pixel shader consume large portion of render budget
- Reuse expensive computation results
  - Reverse reprojection cache (RRC) [Nehab 06, 07]



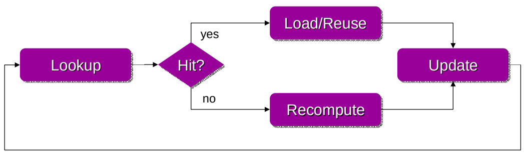
Case study: Pixel shader acceleration

- Regular rendering loop
  - Recompute every pixel using the original pixel shader



Case study: Pixel shader acceleration

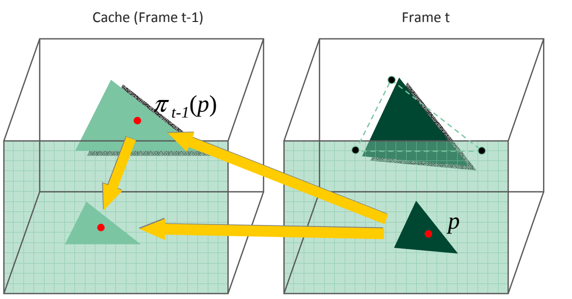
- Reuse previous results using the RRC
  - Reshade on demand
  - Cache reuse path must be cheaper



Outline

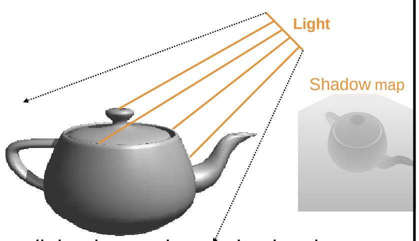
- Image-space spatio-temporal data structure
- Reverse reprojection cache
- Implementation
  - Computing cache coordinate / cache miss
  - Cache resampling
  - Refreshing strategies
  - Control flow
- Determining what to reuse
- Analysis

Determining cache coordinate



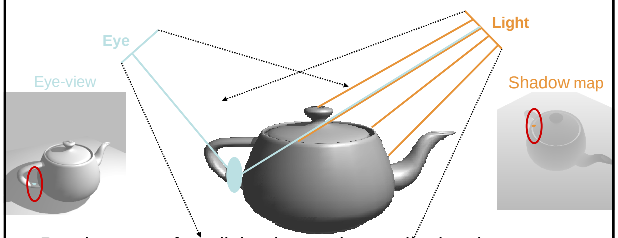
Slide courtesy of Diego Nehab

Analogy: shadow map (first pass)



- Render scene from light-view and save depth values

Analogy: shadow map (second pass)



- Render scene from light-view and save depth values
- Render scene from eye-view
  - Transform each fragment to light source space
  - Compare z_{eye} with z_{light} value stored in shadow map

Determining cache coordinates

- Shader code

Projection space position for *t*-1

Vertex Shader

```
VS_OUTPUT RenderSceneVS(...)
{
   VS_OUTPUT Out;
   // proj-space coordinate for the current frame
   Out.Pos = mul(vPos, g_mWVP);
   // proj-space coordinate for the previous frame
   Out.PosPrev = mul(vPos, g_mWVP_Prev);
   return Output;
}
```

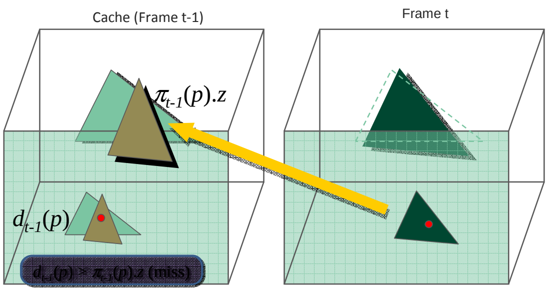
Viewport transform. No need to flip *y* in OpenGL

Pixel Shader

```
float4 RenderScenePS(VS_OUTPUT In)
{
    // perspective division
    In.PosPrev /= In.PosPrev.w;
    // transform coordinates from NDC to screen
    In.PosPrev.xy = (In.PosPrev.xy + 1.f) * 0.5f;
    In.PosPrev.y = 1.f - In.PosPrev.y;
    // fetch the previous value from cache
    float4 cache_val = g_txCache.Sample(
        BiLinearSampler, In.PosPrev.xy);
    ...
}
```

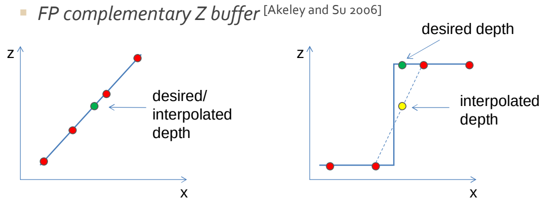

Detecting cache misses

- Depth as an ID



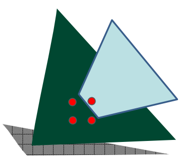
Detecting cache misses

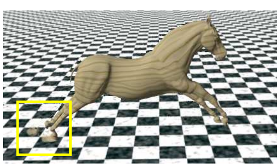
- Bilinear Z interpolation for smooth surface
  - Depth is non-linear but approximate
  - Discontinuity edge: discard
- Z separating threshold ε > depth buffer accuracy
  - *FP complementary Z buffer* [Akeley and Su 2006]



Detecting cache misses

- Intersecting object have similar depths
- Use object ID as an additional ID





Detecting cache misses

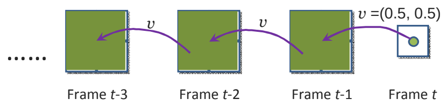
- Viewport clipping
  - Either: invalidate the texture fetch outside the boundary (e.g. Use D3D10_TEXTURE_ADDRESS_BORDER)
  - Or: explicitly test
- Final shader fragment

Pixel Shader

```
bool bHit =
    // clipped case (within [0,1]x[0,1])
    all(In.PosPrev.xy >= 0.0) &&
    all(In.PosPrev.xy <= 1.0) &&
    // occlusion case (depth match)
    abs(In.PosPrev.z-cache_val.w) < g_fZThres;
```

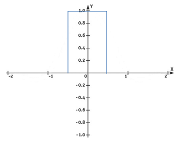
Cache resampling and filtering

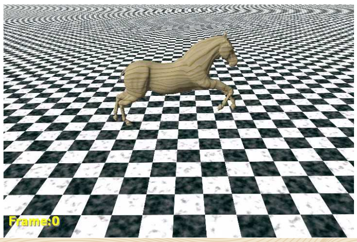
- No 1-to-1 pixel mapping
- Common resampling: Nearest, Bilinear
- Fractional pixel velocity: $v = (v_x, v_y)$



Cache resampling and filtering

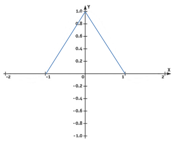
- Nearest (point) resampling
  - Texture shift and distortion

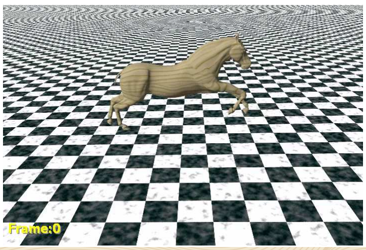




Cache resampling and filtering

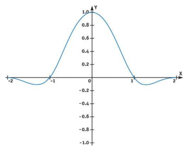
- Bilinear resampling
  - Blur, acceptable < 10 frames





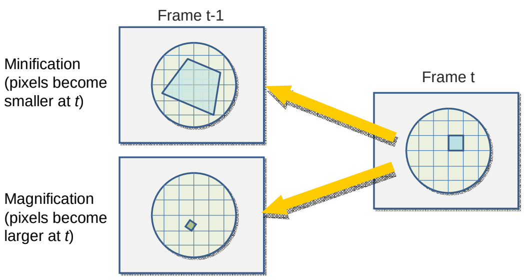
Cache resampling and filtering

- Bicubic resampling
  - Less blur
  - 16 texture fetches can be reduced to 4 [GPU Gems 2, Ch. 20]



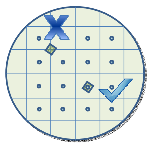
Cache resampling and filtering

- Minification and magnification



Cache resampling and filtering

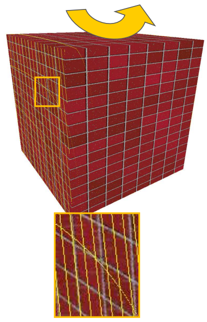
- Minification:
  - Generate a mip chain, read appropriate mip level
- Magnification
  - Estimate error reprojected pixel size and position
  - Force cache miss when reprojected pixel size does not cover any pixel center



Cache resampling and filtering

- Magnification
  - Shader code

```
// "integer" value of the reprojected position
float2 IPrev = In.PosPrev.xy * ScnSz.xy - 0.5;
// reprojected pixel radius
float2 PixR = max(abs(ddx(IPrev)),
                  abs(ddy(IPrev)))*0.5;
// trigger a cache miss if the distance from
// the reprojected position to the nearest
// pixel center is larger than PixR
bool bHit = bHit &&
    all(abs(IPrev - round(IPrev)) <= PixR);
```

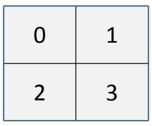


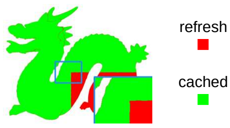
Refreshing strategies

- Source of error
  - Resampling error
  - Shading signal change
- Refresh pixels in round-robin fashion
  - Divide pixels equally into *n* groups
  - Each pixel has a group ID: $i \in [0, n-1]$
  - Refresh when $(t + i) \bmod n = 0$
  - Current frame count: t

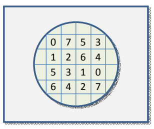
Refreshing strategies

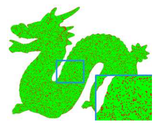
- Tiled refresh





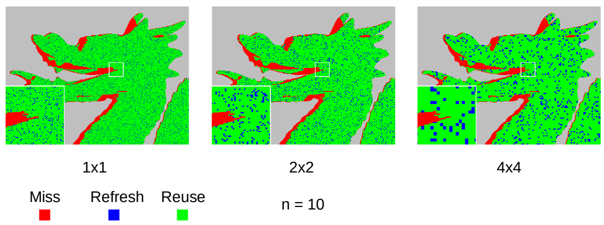
- Random block refresh





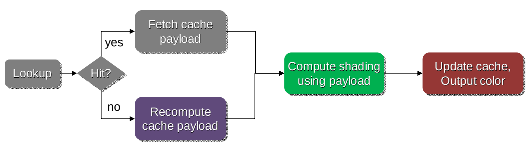
Refreshing strategies

- Random block refresh granularity
  - Block size at least 2 x 2 for GPU efficiency
- Dynamically change *n* per pixel



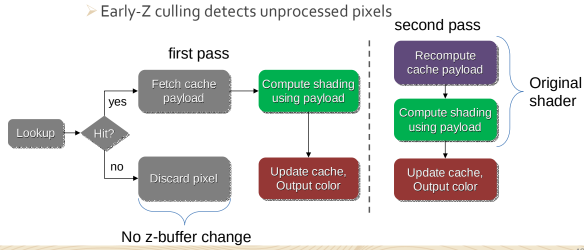
Control flow

- Single-pass implementation
  - Rely on GPU dynamic flow control (DFC)
  - Unbalanced branching causes performance loss
    - Blocks of pixels get penalized by one cache miss



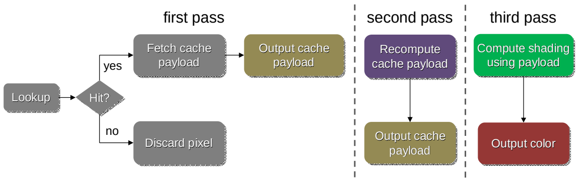
Control flow

- Two-pass implementation
  - First pass: execute cache hit route
  - Second pass: execute cache miss route
    - Early-Z culling detects unprocessed pixels



Control flow

- Three-pass implementation
  - First pass: output cache payload on a hit
  - Second pass: recompute payload on miss pixels (Early-Z)
  - Third pass: Compute the rest of the shading



Outline

- Image-space spatio-temporal data structure
- Reverse reprojection cache
- Implementation
- Determining what to reuse
- Analysis

Determining what to cache

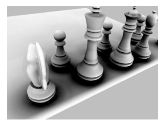
- Reuse arbitrary intermediate computation
- Good candidate
  1. Shading signal changes slowly over time
  2. Weak view- and light-dependency
  3. Expensive to compute
- Maximize saved computational effort relative to caching error

Determining what to cache

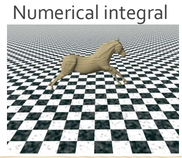
- Good examples to cache

Static procedural texture

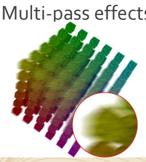
Global illumination approx.



Numerical integral

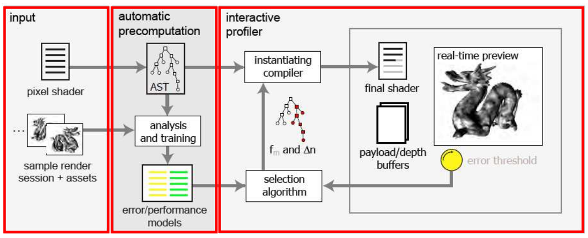


Multi-pass effects



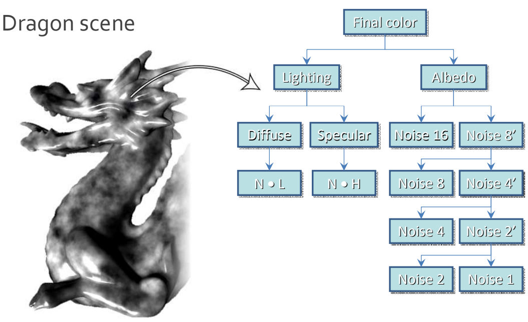
Determining what to cache

- Automatic system [Sitthi-amorn 2008b]
  - Analyze the shader
  - Measure tradeoffs in caching different shading components



Determining what to cache

- Dragon scene

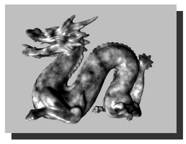


Outline

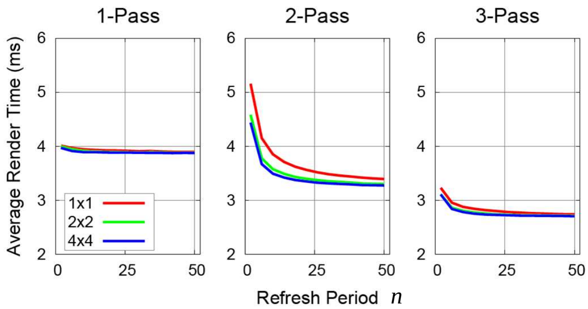
- Image-space spatio-temporal data structure
- Reverse reprojection cache
- Implementation
- Determining what to reuse
- Analysis
  - Performance
  - Quality
  - Quality-speed tradeoff

Performance

- Factors
  - Refreshing strategy
  - Control flow algorithm
  - Cache hit and miss component workload
  - Dynamic branching capability
- Example – dragon scene
  - Single 75k triangle dragon model
  - Perlin-noise albedo, 5 bands
  - Blinn-Phong specular lighting
  - Rotating animation



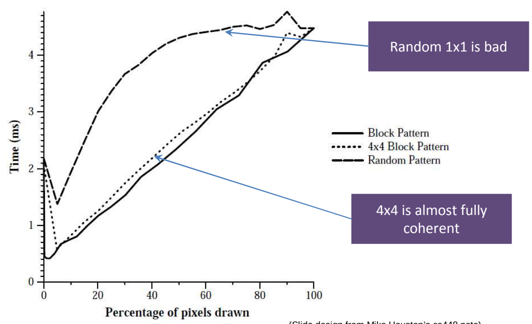
Render time graph [Sitthi-amorn 08a]



Random refresh block size

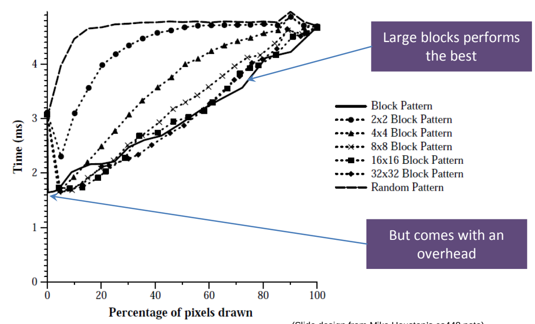
- Performance 1x1 << 2x2 < 4x4
- Early-Z culling granularity: 2x2 pixels
- Dynamic branching granularity:
  - NVIDIA G80 / GT200: 32 pixels
  - AMD Radeon HD5870: 64 pixels

GPU early-Z (2- / 3-pass) efficiency (NVIDIA GTX280)



(Slide design from Mike Houston's cs448 note)

GPU branching efficiency (NVIDIA GTX280)



(Slide design from Mike Houston's cs448 note)

Resampling error [Yang et al. 2009]

- Resampling required when fetching from the cache
- Large $n \rightarrow$ repeated resampling $\rightarrow$ unacceptable blur
- Characterize blur by equivalent Gaussian blur kernel size (or variance)
- The variance of the blur with bilinear resampling: (Fractional pixel velocity: $v = (v_x, v_y)$)

$$\sigma_v^2 = n \cdot \frac{v_x(1-v_x) + v_y(1-v_y)}{2}$$

Grows linearly with n

Maximizes when $v_x = v_y = 0.5$

Summary

- Reverse reprojection cache
  - Light-weight
  - Easy to implement
- Context: shader acceleration
- Performance and quality analysis
- Next:

  Applications of RRC in: Multi-pass effects, amortized sampling, discrete LOD blending, shadows, global illumination, spatial-temporal acceleration