GDC 09

learn
network
inspire

www.GDConf.com

Game Developers Conference®

March 23-27, 2009 | Moscone Center, San Francisco

GSC Game World‘s S.T.A.L.K.E.R : Clear Sky – a showcase for Direct3D 10.0/1

Speakers:
Igor A. Lobanchikov – Former Lead Gfx Engineer at GSC
Holger Gruen - ISV Engineer AMD GPG

Agenda

- » Introduction
- » The X-Ray rendering architecture
- » Notable special effects
- » MSAA deferred rendering 10.0/10.1
- » G-buffer optimization
- » Direct3D 10.1 accelerated effects
- » Q&A

Introduction

- Jon Peddie mentions Stalker : Clear Sky as one of his two top games of 08!
  - JON PEDDIE’S TECH WATCH • Volume 9, NUMBER 1
- The first Direct3D 10.0/1 game to be released with a deferred MSAA renderer
- Contains several Direct3D 10.1 rendering paths
  - MSAA alpha test, accelerated sunshaft and shadows
  - Direct3D 10.1 used for quick prototyping of the MSSA renderer
- This talk walks you through the Direct3D 10.0/1 and other optimizations done in a joint effort between GSC and AMD

The X-Ray rendering architecture

- Rendering stages – list
  - G-stage
  - Light stage
  - Light combine
  - Transparent objects
  - Bloom/exposition
  - Final combine-2
  - Post-effects

The X-Ray rendering architecture: stages

- G-stage
  - Output geometry attributes (albedo, specular, position, normal, ambient occlusion, material).
- Light stage
  - Calculate lighting (diffuse light-RGB, specular light – intensity only)
  - Interleaved rendering with shadowmap
  - Draw emissive objects

The X-Ray rendering architecture: stages

- Light combine
  - Deferred lighting is applied here
  - Hemisphere lighting is calculated here (both using OA light-map and SSAO)
  - Perform tone-mapping here
  - Output Hi and Lo part of tone-mapped image into 2 RTs

The X-Ray rendering architecture: stages

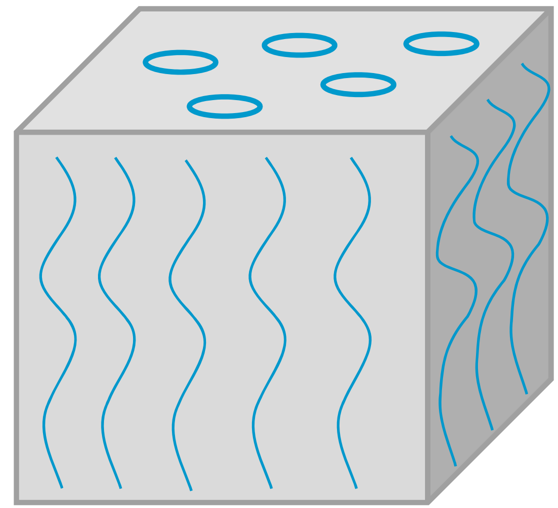
- Transparent objects
  - Basic forward rendering
- Bloom/exposition
  - Use Hi RT as a source for bloom/luminance estimation
- Final combine-2
  - Apply DOF, distortion, bloom here
- Post-effects
  - Apply black-outs, film-grain, etc..

Dynamic rain

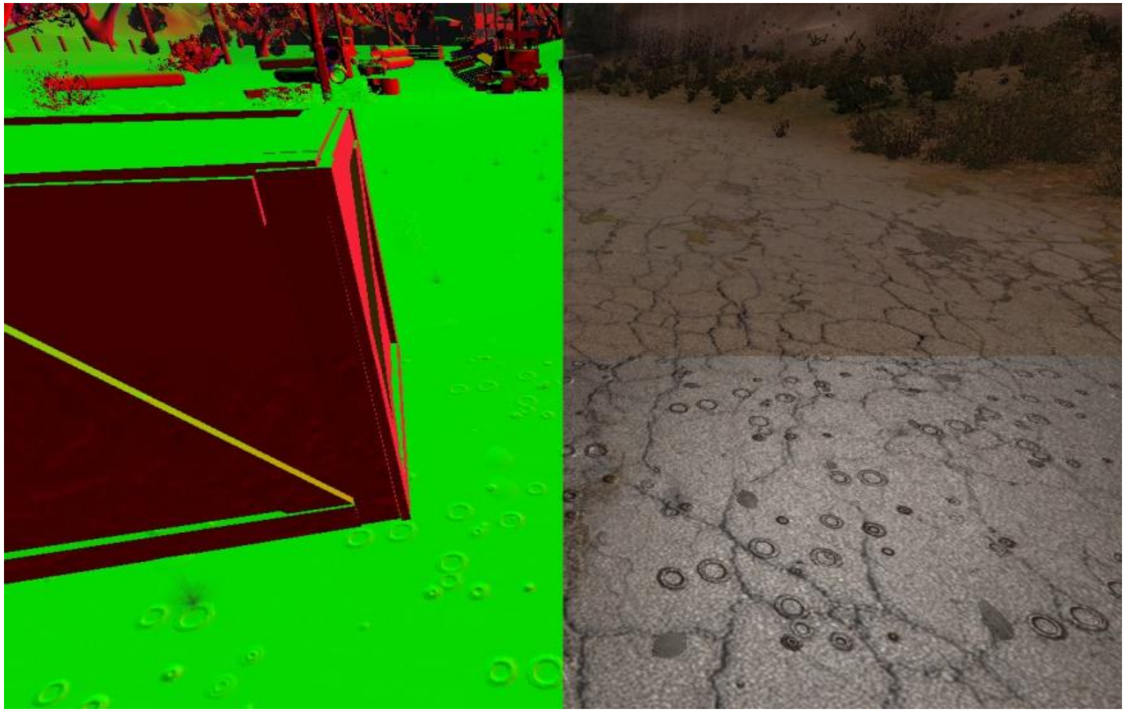
- Prepare shadowmap as seen along the direction of rain
  - Visible pixels are considered wet
- Apply postrpocess to G-buffer
  - Make albedo darker and specular higher
  - Fix-up normal
- That's all

Dynamic rain: normal fix-up

- Horizontal surfaces
  - Use tiled volume texture to animate puddle rings
- Vertical surfaces
  - Scroll texture with the water stream vertically
- All normals are treated as world-space ones



Dynamic rain: G-buffer modification



Normal visualization Combined image

Dynamic rain disabled

Dynamic rain enabled

Dynamic rain: shadowmap

- Use shadowmap to mask pixels invisible to the rain
  - Draw only static geometry
  - Snap shadowmap texels to world space
  - Use jittering to hide shadowmap aliasing and simulate wet/dry area border.

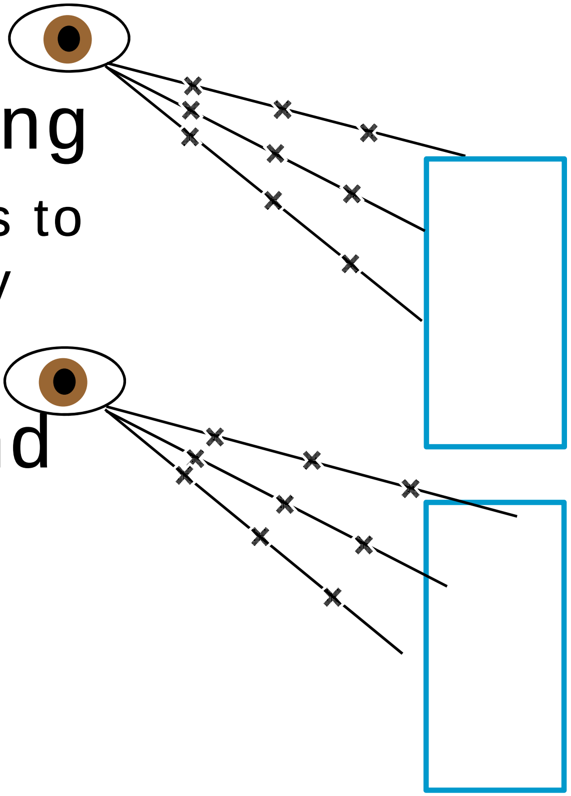
Dynamic rain: shadowmap

4-tap shadowmap Jittered shadowmap

Dynamic rain: what's next?

- Use material ID
- Use more directions for gradient detection
- Puddle map
  - Project additional puddle textures on the ground for artist-defined behavior
- Use reprojection cache?
  - Storing rain shadowmap history from the previous frame could allow us to use dynamic objects as rain occluders

Sun Shafts



- Just do ray-marching
  - Shadowmap test needs to be carried out on every step
- Jitter ray length and use PCF to hide banding artifacts
- Use lower single sample intensity to hide noise

Sun Shafts performance considerations

- High sampling shadowmap coherency due to the high coherency of positions in G-buffer (breaks for A-tested geometry)
- Even higher sampling coherency for dueling frustum case
- Fixed number of steps eliminates dynamic branching which helps in low coherency edge cases

Sun Shafts: Cascaded Shadow Map case

- Just use single cascade for the whole ray
  - Simpler algorithm
  - Lower resolution shadowmap reduces banding for longer rays
  - Visible border between cascades

MSAA deferred rendering 10.0/10.1

- Deferred MSAA Rendering under dx10
  - main concept
  - stages affected by MSAA rendering
  - Easy prototyping with Direct3D 10.1
  - dx10 A2C

MSAA deferred rendering 10.0/10.1 main concept

» Render to MSAA G-buffer.

» Mask edge pixels.

» Process only subsample #0 for plain pixels. Output to all subsamples.

» Process each subsample for edge pixels independently.

MSAA deferred rendering: MSAA output

- **G-stage** (subsample geometry data)
- **Light stage** (subsample lighting)
- **Light combine** (subsample data combination)
- **Transparent objects**
- Bloom/exposition
- Final combine-2
- Post-effects

MSAA deferred rendering: read from MSAA source

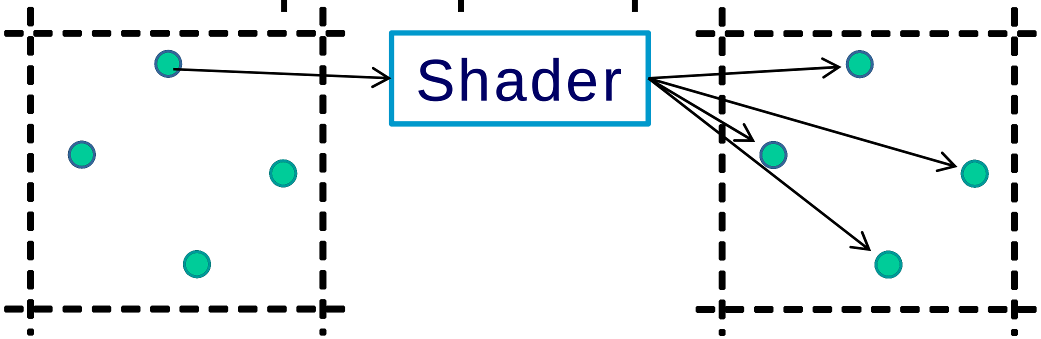
- G-stage
- **Light stage** (uses G-stage data)
- **Light combine** (uses G-stage and light stage data)
- Transparent objects
- Bloom/exposition
- Final combine-2
- Post-effects

MSAA deferred rendering: MSAA in/out stages

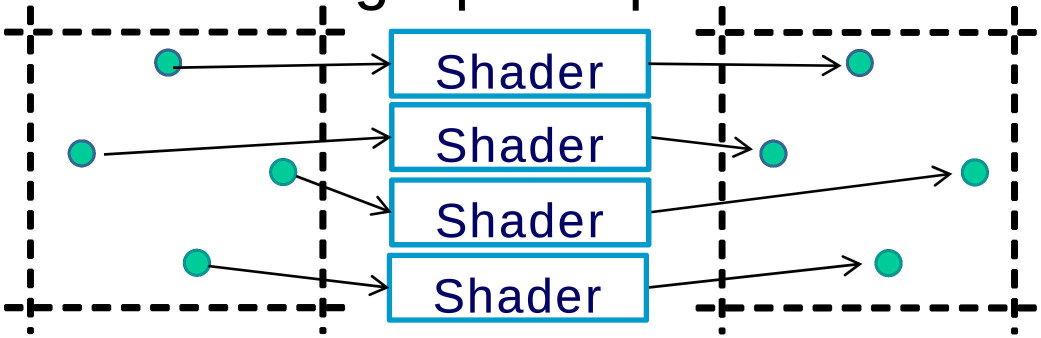
- For each shader
  - Plain pixel – run shader at pixel frequency
  - Edge pixel – run at subpixel frequency
- Early stencil hardware minimizes PS overhead

MSAA deferred rendering: MSAA in/out stages

plain pixel pass



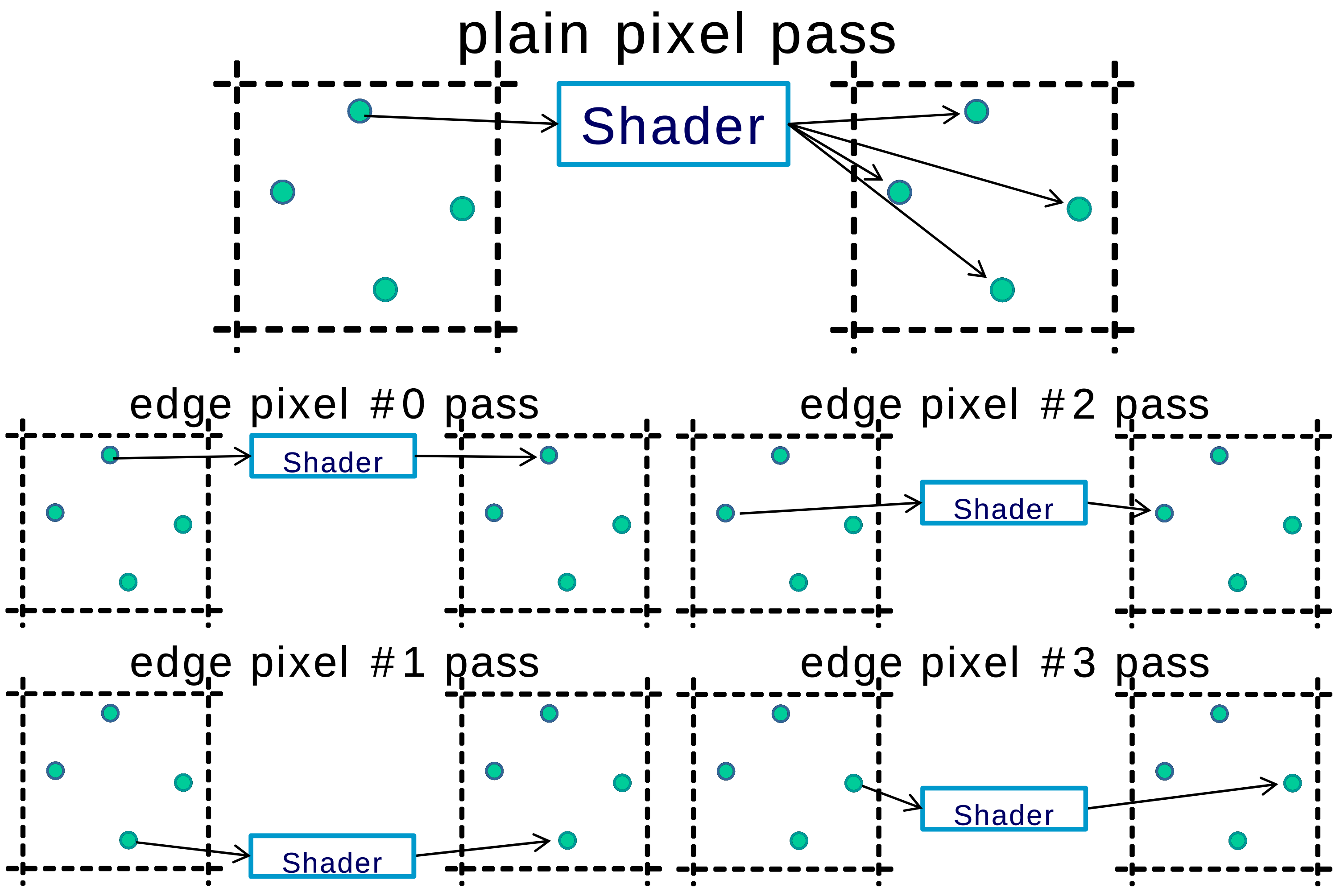
edge pixel pass



MSAA deferred rendering

- DX10 doesn‘t support running shader at subsample frequency (DX10.1 does).
- Use DX10.1 for fast prototyping.
- For DX10 use separate pass
  for each subsample: shaders specifies subsample to read at compile time, use output mask to allow writing to a single subsample.

MSAA deferred rendering: DX10



MSAA deferred rendering: DX10 A2C

- A-tested geometry can‘t be MSAA‘d using common technique.
- Use A2C to approximate anti-aliasing.
- Alpha-channel of all g-buffers store geometry attributes: need 2-pass algorythm:
  - Write only depth using A2C
  - Write geometry data using Z-equal.

G-buffer optimization - 1

- Stalker originally used a 3-RT G-buffer
  - 3d Pos + materialID => RGBA16F RT0
  - Normal + Ambient occl. => RGBA16F RT1
  - Color + Gloss => RGBA8 RT2
- At high resolutions/msaa-settings the size of the G-buffer becomes the bottleneck
- Joint effort optimization effort lead to a 2-RT G-buffer
  - Normal+Depth+matID+AO => RGBA16F RT0
  - Color + Gloss => RGBA8 RT1
  - Trade packing math vs. less g-buffer texture ops
  - Reduces G-buffer size from 160 to 96 bits pp

G-buffer optimization - 2

» Reconstruct 3d position from depth

```
// input SV_POSITION as pos2d
New_pos2d = ( (pos2d.xy) * (2/screenres.xy) )– float2(1,1);
viewSpacePos.x =  gbuffer_depth  *  tan( 90-HORZFOV/2 ) * New_pos2d.x;
viewSpacePos.y = -gbuffer_depth  *  tan( 90-VERTFOV/2 ) * New_pos2d.y;
viewSpacePos.z =  gbuffer_depth;
```

» Normals get un-/packed from 2d <-> 3d

» Packing

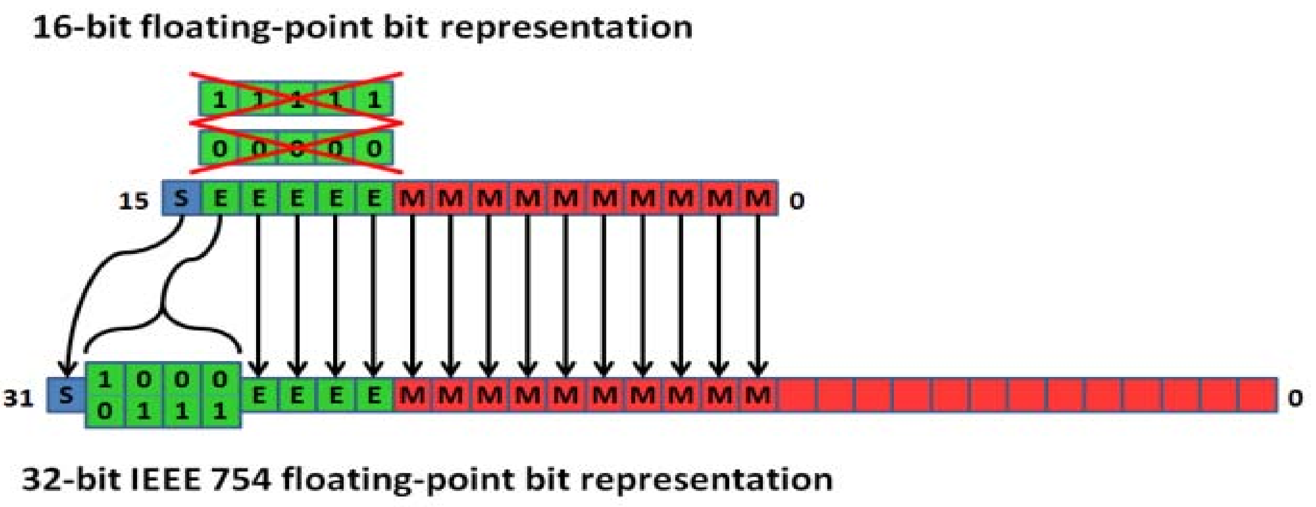
```
float2 pack_normal( float3 norm )
{
  float2 res;
  res = 0.5 *( norm.xy +
               float2( 1, 1 ) ) ;
  res.x *= ( norm.z < 0 ? -1.0 :
                     1.0 );
  return res;
}
```

» Unpacking

```
float3 unpack_normal(float2 norm)
{
    float3 res;
    res.xy= ( 2.0 * abs( norm ) ) –
             float2(1,1);
    res.z =(norm.x < 0? -1.0:1.0)*
           sqrt( abs( 1 –
                  res.x*res.x-
                  res.y*res.y));
    return res;
}
```

G-buffer optimization - 2

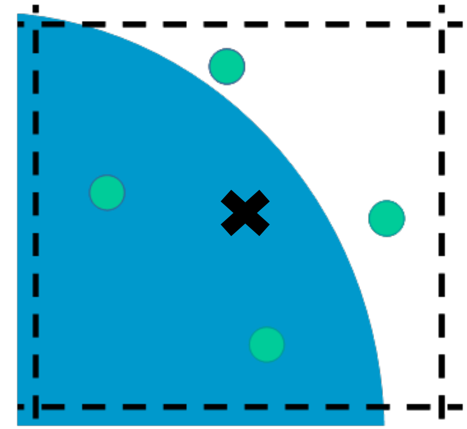
- pack AO and matID into the usable bits of the last 16bit fp channel of RT0
  - Pack data into a 32bit uint as a bit pattern that is a valid 16bit fp number
  - Cast the uint to float using asfloat()
  - Cast back for unpacking using asuint()
  - Extract bits



Direct3D 10.1 accelerated effects - Agenda

- MSAA Alpha test
  - A brief recap
- Shader based A2C
  - Why would you want to do this in a shader?
- Non-hierarchical min-max shadow maps
  - Hybrid plane based/min-max solution
- Direct3D 10.1 accelerated shadows
  - A teaser for the upcoming talk from Jon and I

Direct3D 10.1 accelerated effects – MSAA Alpha Test



- Sample texture once for each MSAA sub-sample
  - ddx/ddy used to find UV coordinates at sub-samples
  - Sample locations standardized in Direct3D 10.1
- Set SV_COVERAGE for samples passing the AT
- Higher image quality than Direct3D 10.0 A2C!
- One rendering pass only in Stalker
  - A2C need two passes in Stalker under Direct3D 10.0
  - => good for CPU limited situations in Stalker
- More texture-heavy than Direct3D 10.0 A2C especially at 8xmsaa


Settings
Benchmark
Free camera
Exit
00:00
12:53
24:00


Settings
Benchmark
Free camera
Exit
00:00
12:53
24:00

Direct3D 10.1 accelerated effects – Shader A2C

- Why would you want to do this?
  - MSAA Alpha test slower than A2C at high (msaa) settings
  - Control over SV_COVERAGE allows one-pass-shader based A2C in Stalker
    - Direct3D 10.0 A2C needs two passes in Stalker
  - Shader based A2C only needs to look at one texture sample
  - Admittedly lower quality than MSAA AT but sometimes speed is all you care about

Direct3D 10.1 accelerated effects – Shader A2C cont.

Tried two methods to implement this.

<u>Method 1 at 4xMSAA</u>

$$\alpha \begin{cases} \xrightarrow{\alpha < 1/5} & \text{0 bits in SV\_COVERAGE set} \\ \xrightarrow{1/5 \le \alpha < 2/5} & \text{1 bit in SV\_COVERAGE set} \\ \xrightarrow{2/5 \le \alpha < 3/5} & \text{2 bits in SV\_COVERAGE set} \\ \xrightarrow{3/5 \le \alpha < 4/5} & \text{3 bits in SV\_COVERAGE set} \\ \xrightarrow{4/5 \le \alpha \le 5/5} & \text{4 bits in SV\_COVERAGE set} \end{cases}$$

(1 to 3 bits:) SV_POSITION.x + SV_POSITION.y used to select bit pattern

Direct3D 10.1 accelerated effects – Shader A2C cont.

<u>Method 2 at 4xMSAA</u>

$$\alpha \xrightarrow{\substack{\text{SV\_POSITION.x+} \\ \text{SV\_POSITION.y} \\ \text{used to perturb} \\ \text{alpha}}} \alpha' \begin{cases} \xrightarrow{\alpha' < 1/5} \text{SV\_COVERAGE} = 0 \\ \xrightarrow{1/5 \le \alpha' < 2/5} \text{SV\_COVERAGE} = 1 \\ \xrightarrow{2/5 \le \alpha' < 3/5} \text{SV\_COVERAGE} = 3 \\ \xrightarrow{3/5 \le \alpha' < 4/5} \text{SV\_COVERAGE} = 7 \\ \xrightarrow{4/5 \le \alpha' \le 5/5} \text{SV\_COVERAGE} = 15 \end{cases}$$

This method got used in Stalker –
it is simply cheaper!

Direct3D 10.1 accelerated effects – Shader A2C cont.

No obvious difference in IQ expected – only a zoom-in shows a difference

Direct3D 10.0 A2C

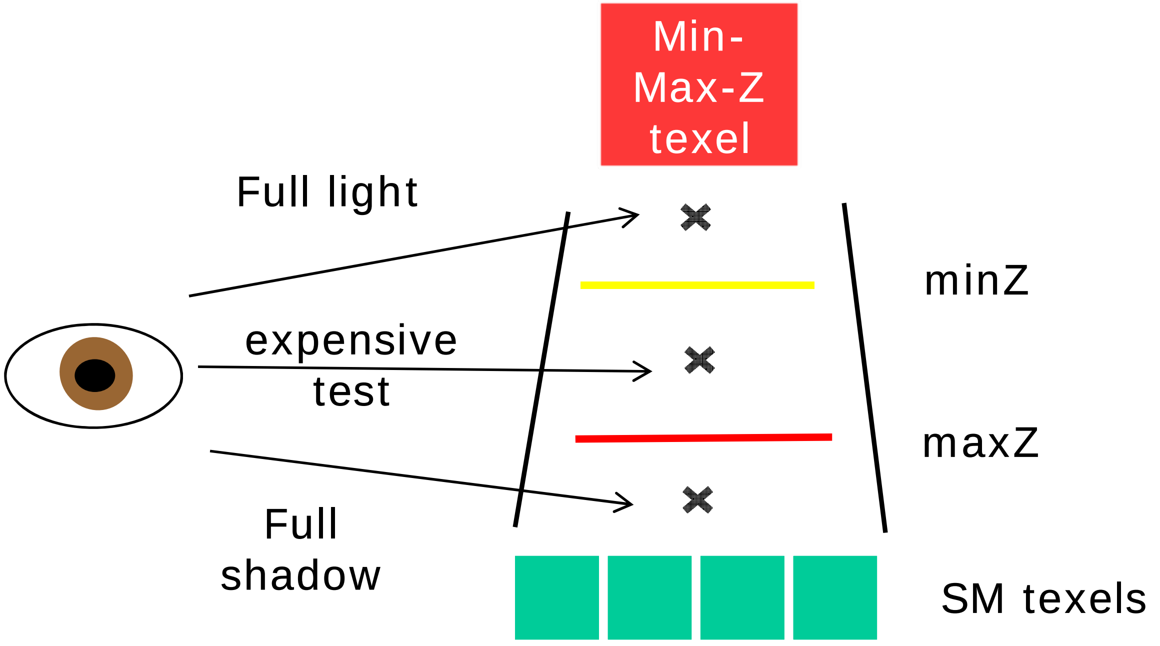
Direct3D 10.1 shader based A2C

Direct3D 10.1 accelerated effects – min-max SM Recap

- Min-Max Shadows Maps introduced at GDC 05 by K. Dmitriev & Y. Uralsky
- Key idea: Build mip-chain from a shadow map
  - Store min/max depth of 4 texels in next mip down
- Allows hierarchical rejection of sub-blocks of a shadow filter kernel
  - Traverse mips and check for overlap of shadow filter kernel quad with current min-max SM quad
  - If center.z >= max Z => full shadow
  - Else if center.z < min Z => full light
  - Can accelerate large filter kernels

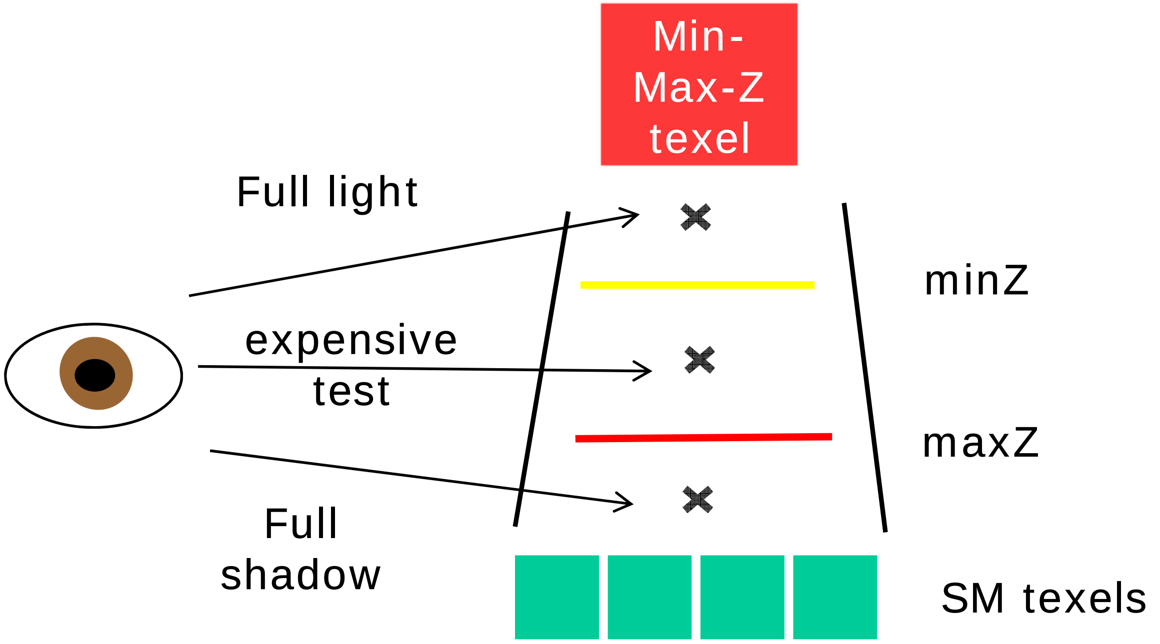
Direct3D 10.1 acc. effects – non-hierarchical min-max SMs

» Reduce NxN block of SM texels to one min-max texel



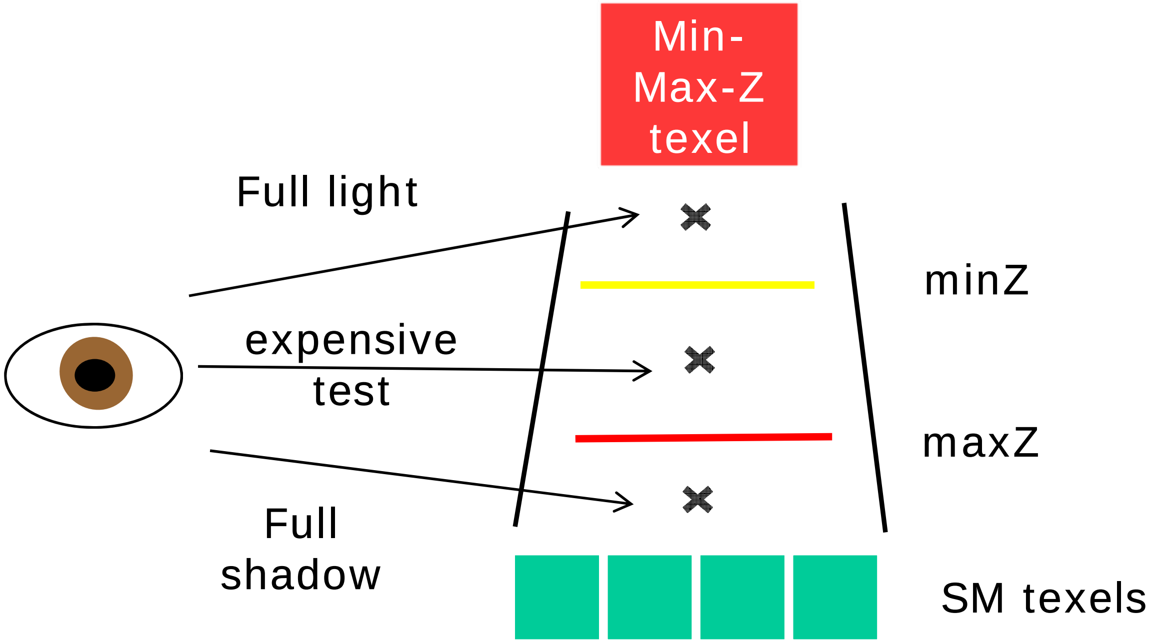
Direct3D 10.1 acc. effects – non-hierarchical min-max SMs

» Can still be used to reject sub-blocks of a shadow filter kernel



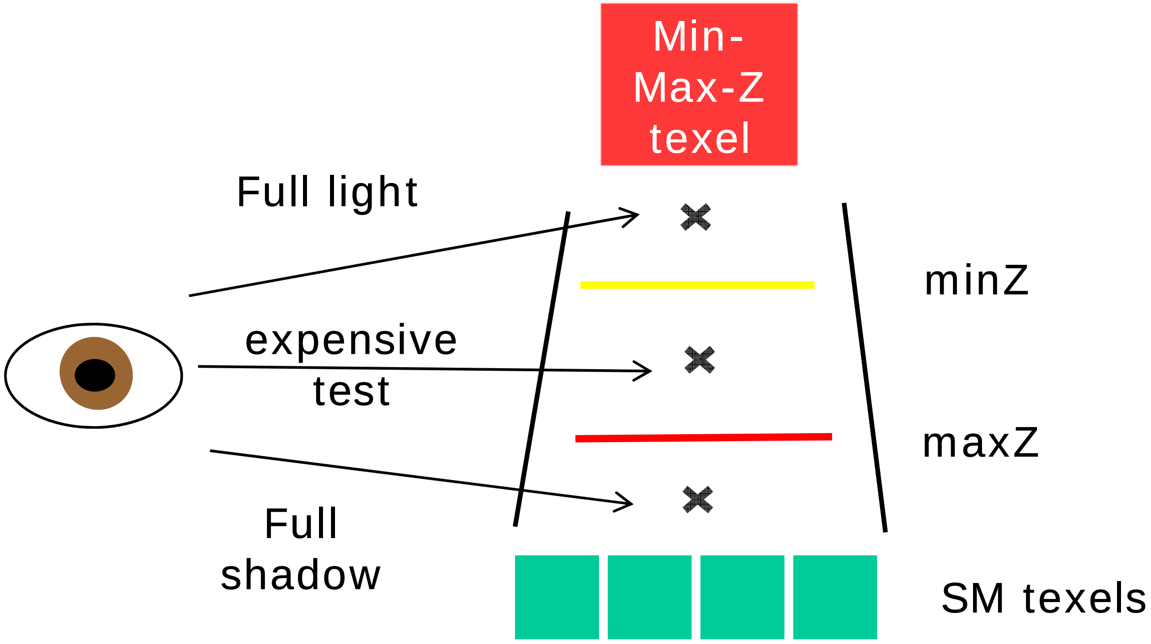
Direct3D 10.1 acc. effects – non-hierarchical min-max SMs

» Same test logic as hierachical min-max SMs



Direct3D 10.1 acc. effects – non-hierarchical min-max SMs

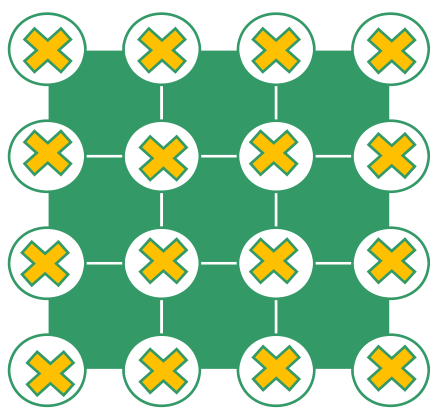
- Higher chance for one-sample quick rejection test than hierachical min-max SM



Direct3D 10.1 acc. effects – min-max SM construction

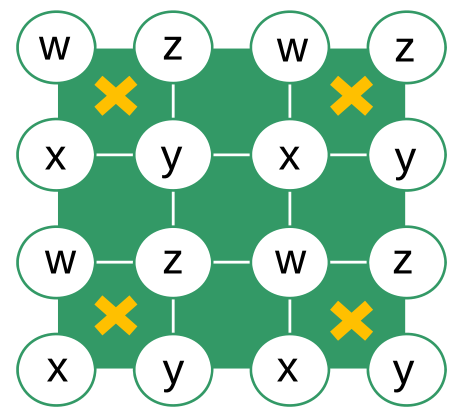
Direct3D 10.1 accelerates min-max SM construction – e.g. for a 4x4 to 1 reduction

Direct3D 10.0



NxN (4x4) = 16 point samples if one wants to find min-max depth

Direct3D 10.1



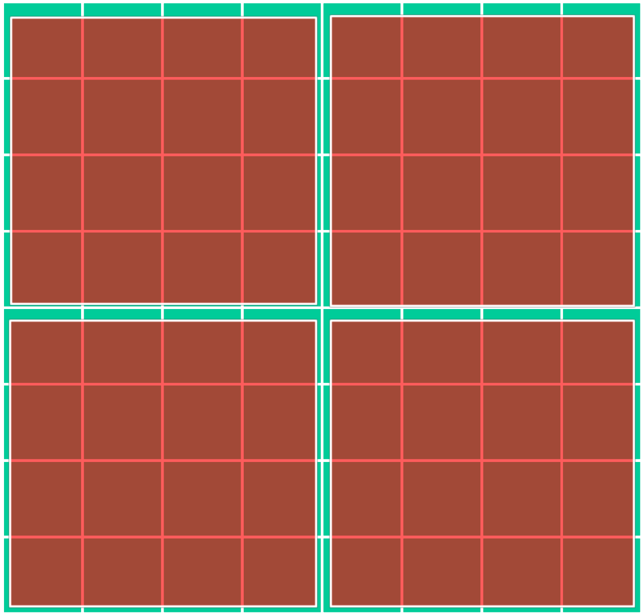
(N/2)x(N/2) = 4 Gather() samples get all data

Direct3D 10.1 acc. effects – non-hier. min-max SMs cont.

Things to consider when using the min-max SM ..

Let's consider a shadow map

And its min-max SM

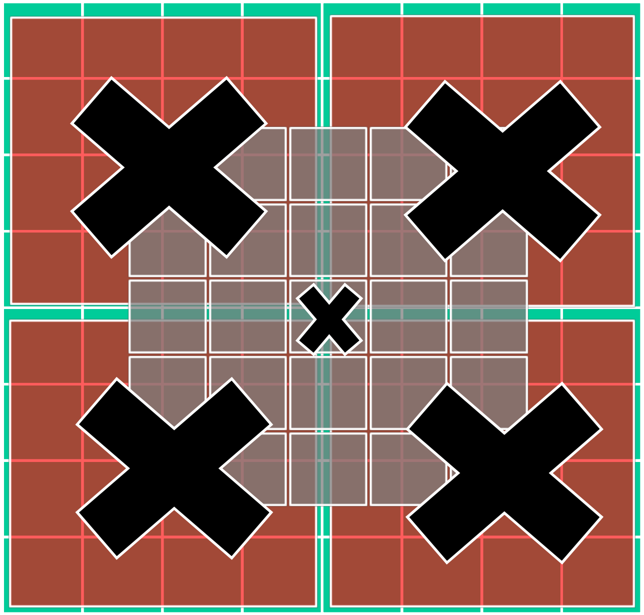


Direct3D 10.1 acc. effects – non-hier. min-max SMs cont.

Things to consider when using the min-max SM ..

A shadow mapping filter kernel can overlap four min-max SM texels

It is necessary to sample all min-max texels that are touched by the kernel

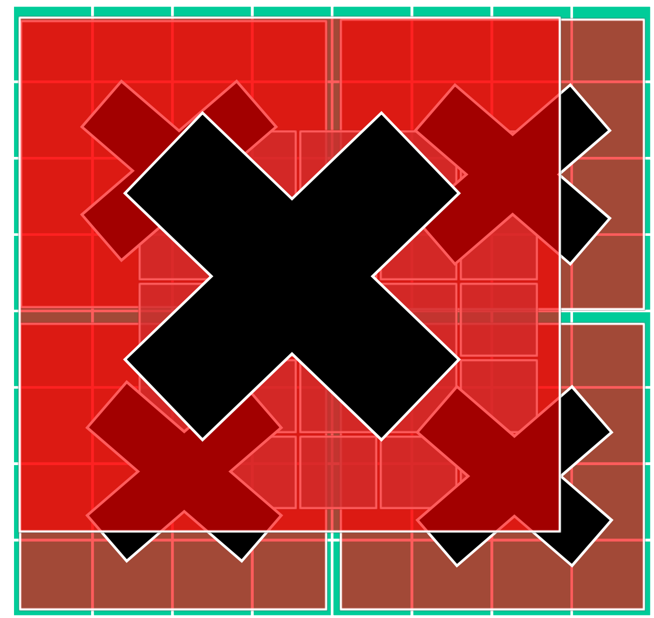


Direct3D 10.1 acc. effects – non-hier. min-max SMs cont.

Things to consider when using the min-max SM ..

Instead one can just have overlapping min-max construction kernels

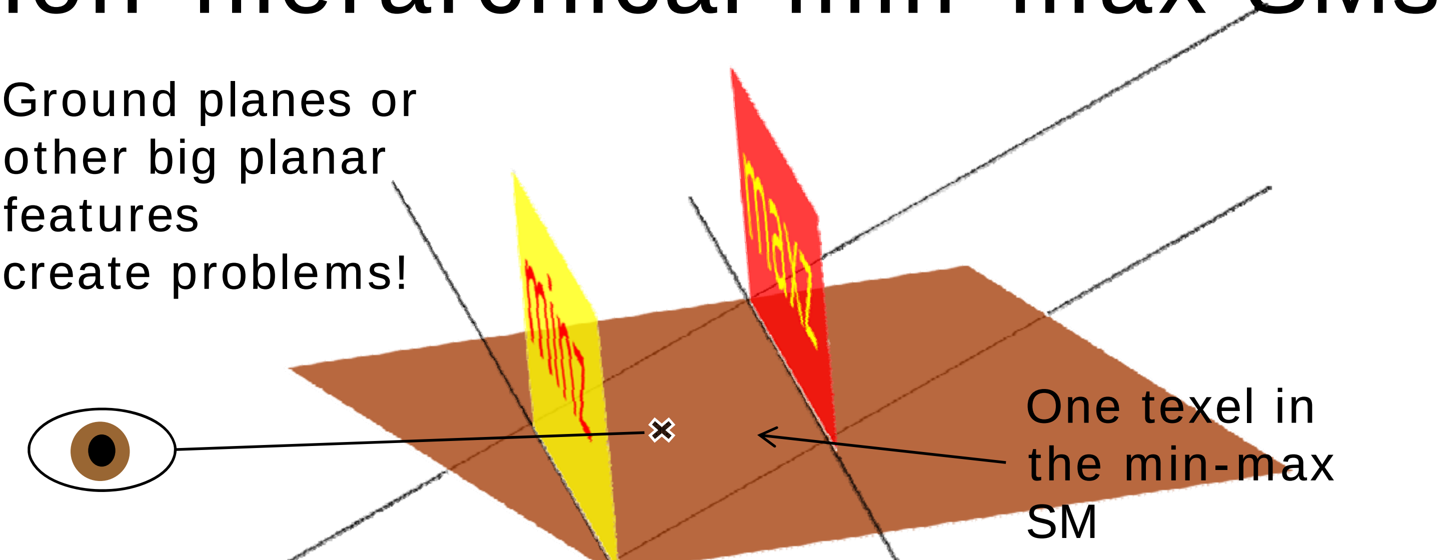
And use only one sample !



Stalker uses an overlapping filter kernel big enough to allow quick rejection of sunshaft shadow map probes and uses Gather() to accelerate min-max SM lookups for big shadow filter kernels

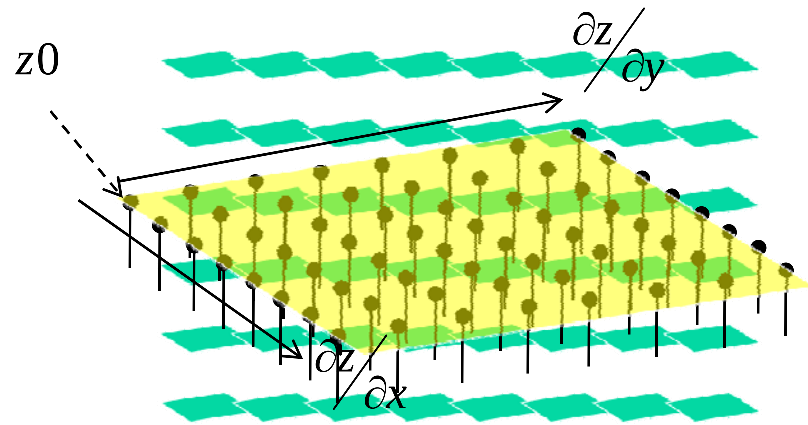
Direct3D 10.1 acc. effects – non-hierarchical min-max SMs

Ground planes or other big planar features create problems!



- All pixels within the projected area are between minZ/maxZ
  - All go down the expensive path
- Only way around this is a high depth bias with all its problems
- Too low depth bias => bad DFC coherency
  - Also an issue for hierachical min-max SMs

Direct3D 10.1 acc. effects – non-hier. min-max SMs cont.



- Try to fit a plane through all depth samples in the filter kernel
- Store plane equation as RT0:-z0, RT1:(dz/dx,dz/dy) instead of RT0:maxZ, RT1: minZ, 0)
- shadow shader uses sub-texel offsets to compute depth D using stored plane equation
  - D used as minZ and maxZ
- This solves the issues with planar features!
  - Save to assume that the whole filter kernel is in front of or behind the plane

Direct3D 10.1 acc. effects – non-hier. min-max SMs cont.

- Why use min-max SMs in Stalker ?
  - Allows for a higher shadow quality at high FPS
    - Rejects most pixels as fully lit or fully shadowed
    - Expensive 8x8 shadow filter only for pixels that need them
  - Min-Max SMs accelerate sunshaft rendering
    - Each step needs to do up to 4 PCF lookups into the shadow map per step on the ray
    - Uses min-max SM to skip these lookups if possible

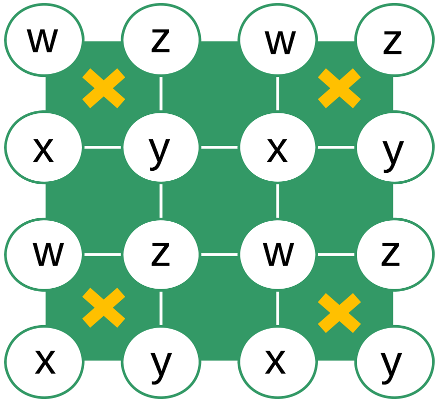

22


22

Direct3D 10.1 accelerated shadows - teaser

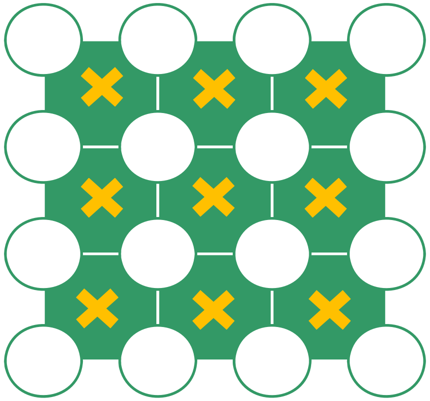
Let’s filter a 4x4 visibility sample block for smooth shadows

Direct3D 10.1



4 Gather() samples plus some ALU => (N/2)x(N/2) Gather() samples for NxN

Direct3D 10.0



9 PCF samples plus some ALU right?

Q&A

Please Fill in the Feedback Forms…