Intel® 4th Generation Core DX11 Extensions

Getting Kick Ass Visual Quality out of the Latest Intel GPUs

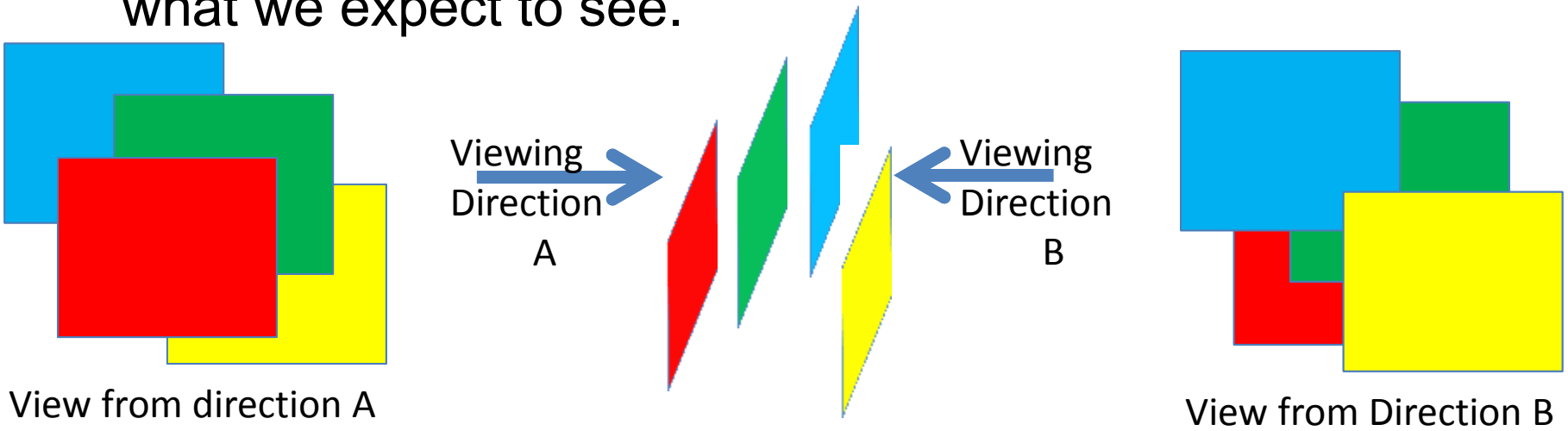
Steve Hughes: Senior Application Engineer - Intel

The unspoken problems in Graphics

- How do I sort alpha polygons quickly?
- How do I sort alpha polygons correctly?
- How do I achieve programmable blending in DirectX?
- How do I transfer data to and from GPU memory efficiently?

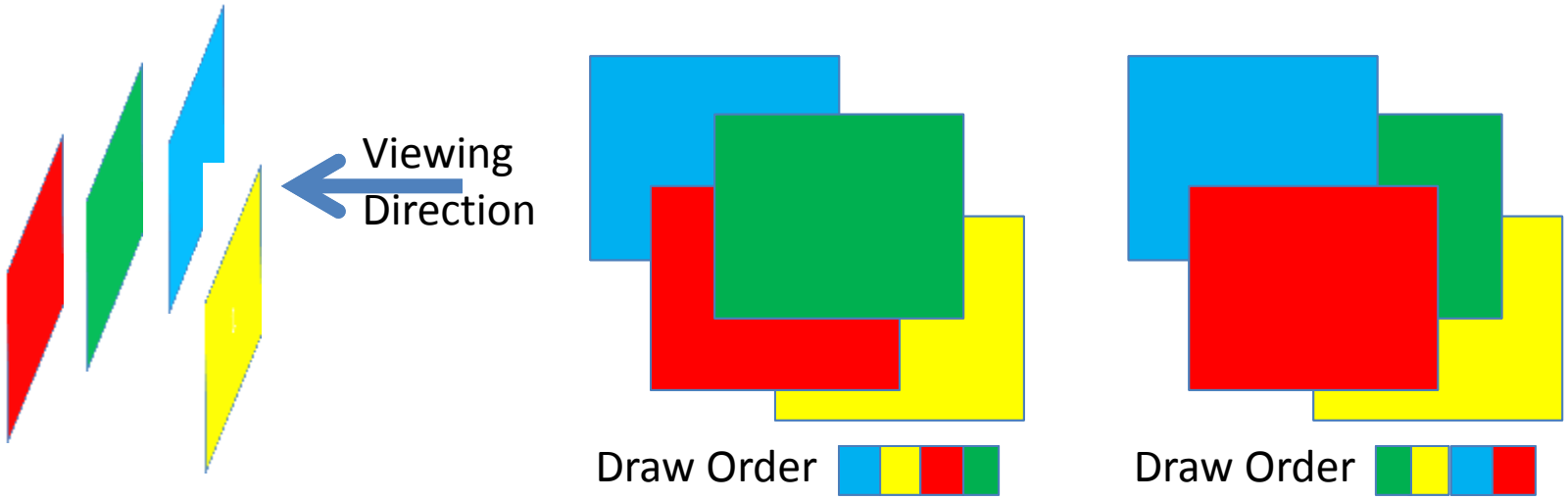
Why do I mean by alpha polygons?

- Lets imagine in our game we have these 4 polygons. They are in our 3D scene, and this is what we expect to see.



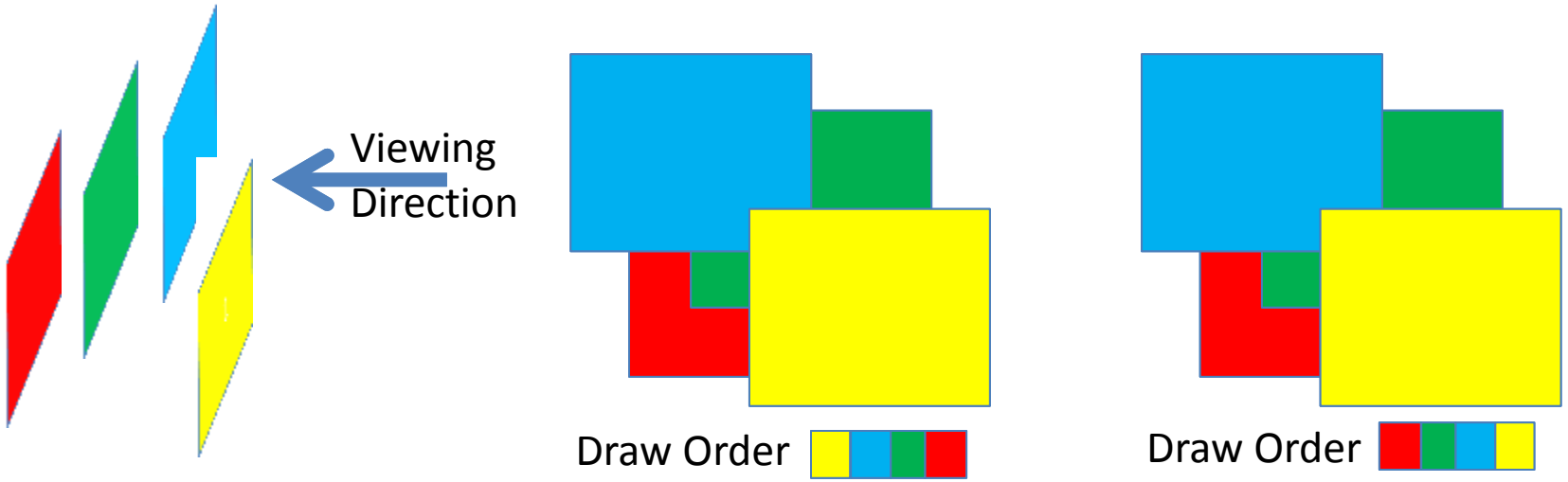
How do we make that happen?

- They have to be applied to the screen in back to front order otherwise…



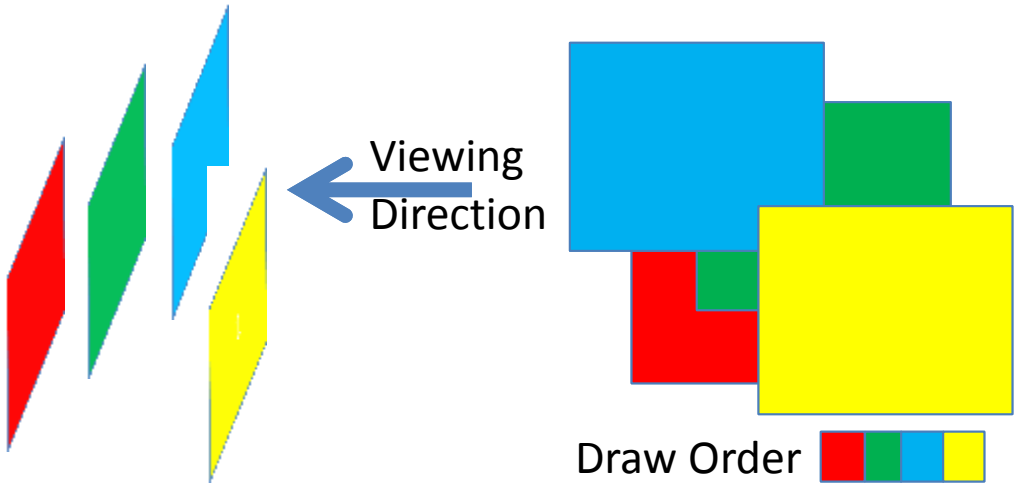
What about the z-buffer?

- If you use the z-buffer you don't get all your pixels…



Is there a right way to do this at all?

- Yes, you sort them back to front and draw them without z…after all of the opaque polygons in the scene.



"Sort" is the key word…

Ok, so they look right - sorting 4 polygons isn't too expensive though, is it?

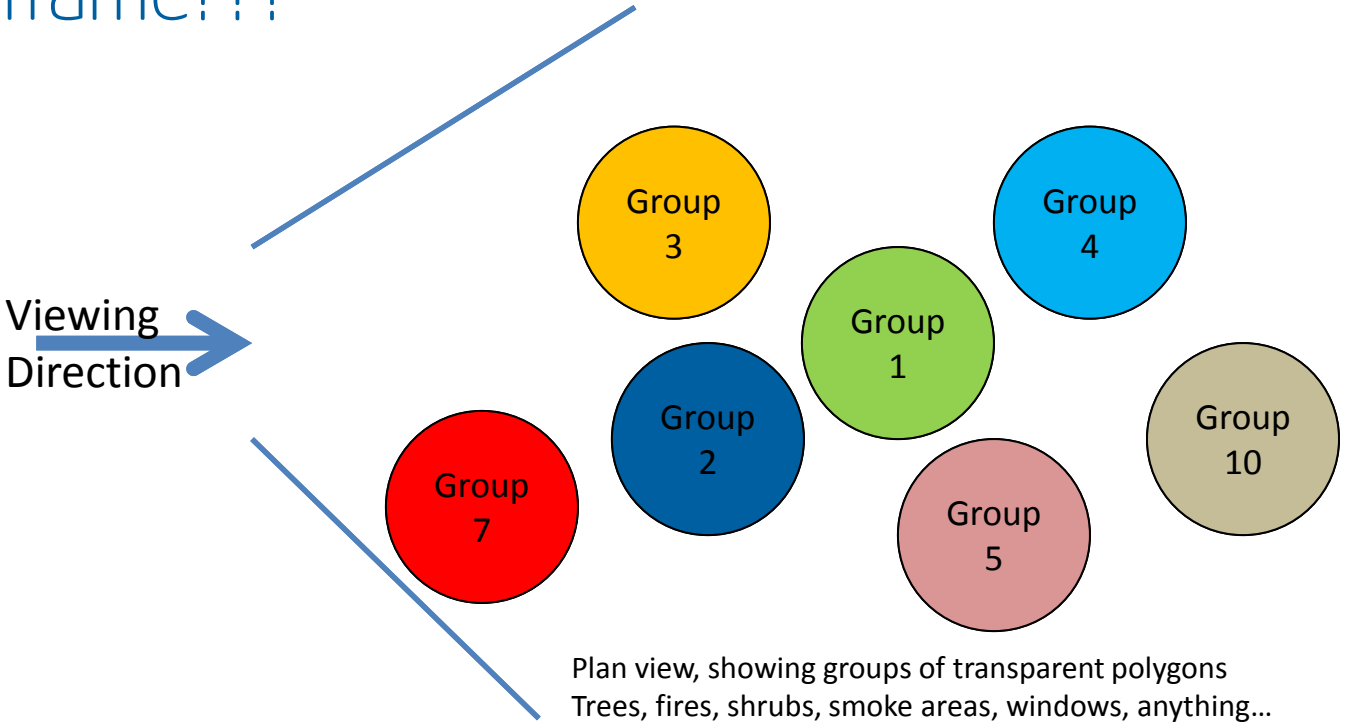
A more realistic example!

High poly count tree:

- All leaf models are in the same vertex buffer
- All leaf edges have Alpha values
- Tree is animated
- Need to re sort every frame to draw correctly

Very expensive to get right!

So, do we have to sort millions of poly's every frame???

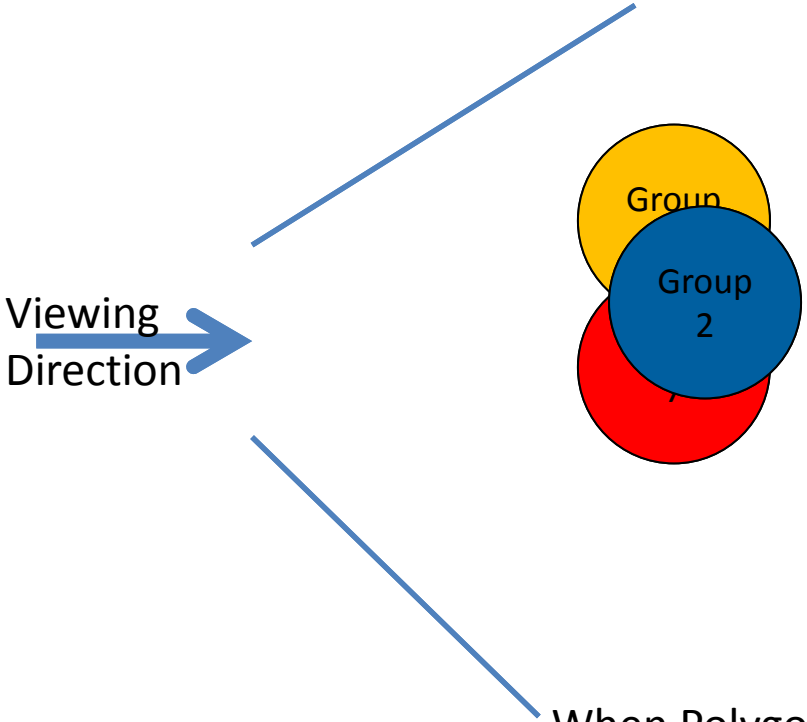


Plan view, showing groups of transparent polygons
Trees, fires, shrubs, smoke areas, windows, anything…

Well, no.
Poly's split into groups
Sorted within the group
Then sort the groups…

Does that work?

And, does it work?...

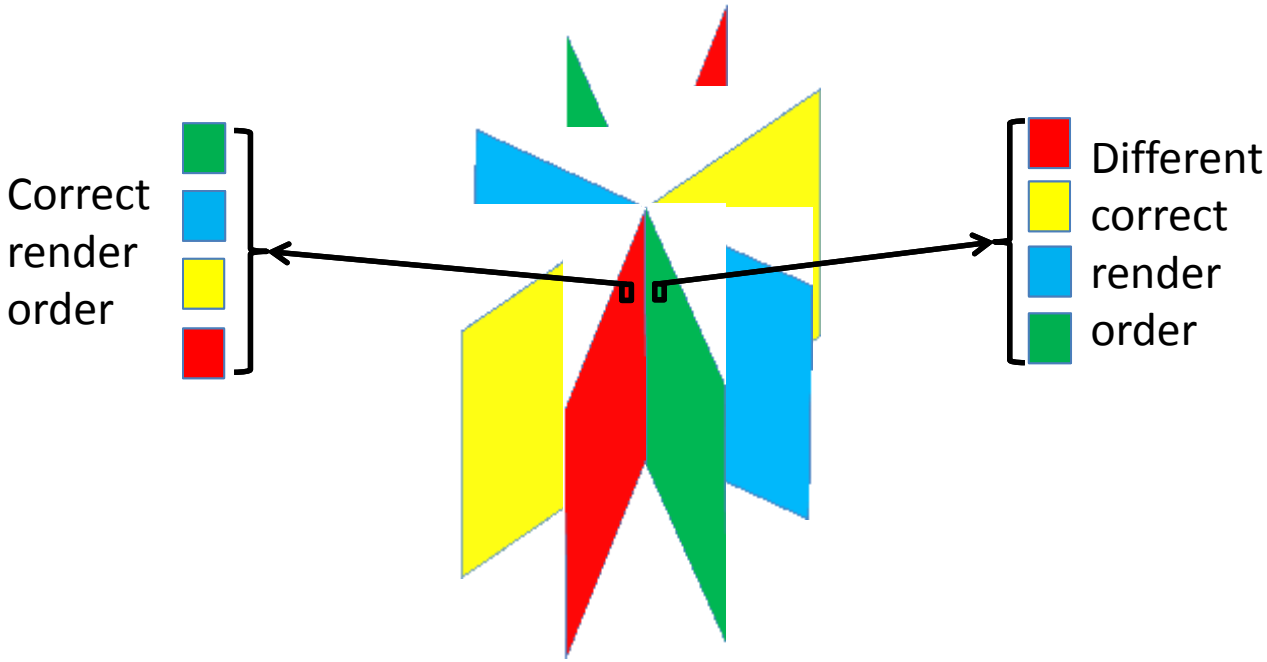


Well, no.
It does as long as you take steps to keep groups apart.

2 things go wrong here:
- One group will always draw over another
- Polygons will intersect others…

When Polygons Intersect… (Discovery Channel, Friday, 9pm)

A closer look at intersecting polygons



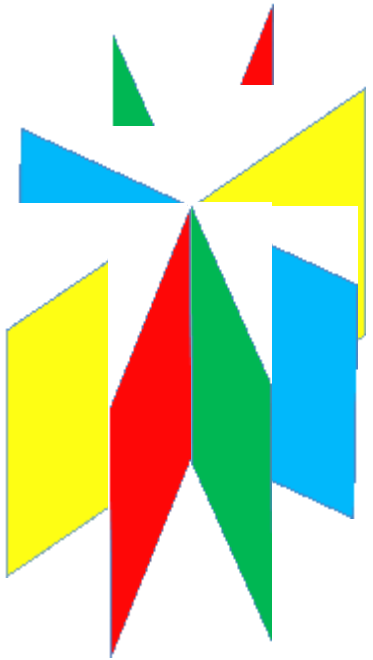
No sort plan will ever let you sort the pixels in these 4 polygons correctly!

Does that ever happen, really?

Go on, ask an artist!

Pause for thought…

Scene sorting doesn't work.
Its also expensive.
This is an old, old problem.



Can anything help…
How about Unordered Access Views (UAVs) and Structured Buffers?

How about Unordered Access Views and Structured Buffers?

What are they?

Structured Buffer

- New resource type to DX11.
- Each entry is a fixed size structure defined by the user.

Unordered Access View (UAV)

- View of a buffer which allows scatter / gather operations – hence u*nordered*
- Shader can write to any location.

How does that help us?

2 pass proposition for solving the alpha sort problem:

Pass 1 pixel shader

Stores first ‘n’ color/alpha/z triplets in the buffer.
Stores remaining colors by merging the triplet into one of first ‘n’ colors as appropriate.

Pixel 123, 67

Buffer data
Per screen loc

RGB	A	Z
	0.7	0.1
	0.3	0.2
	0.9	0.6
	0.5	0.8

Screen loc 123, 67
In buffer

Pass 2 pixel shader

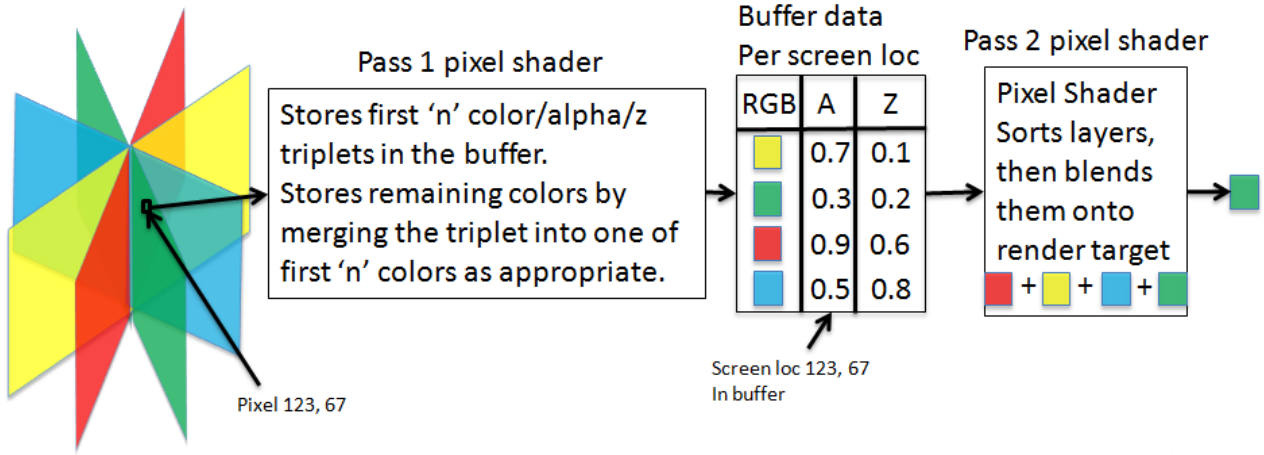
Pixel Shader
Sorts layers, then blends them onto render target

+ + +

After the first ‘n’ colors, successive colors are adaptively merged into one of the first ‘n’. Great example of how to do this here:
http://software.intel.com/en-us/articles/adaptive-transparency

You can’t do this with DX11 alone!

Why won't that work?:



1. Pixels arrive in random order (it's a UAV, right?) after the first 4, pixels can be merged into the buffer in an inconsistent order on successive frames.

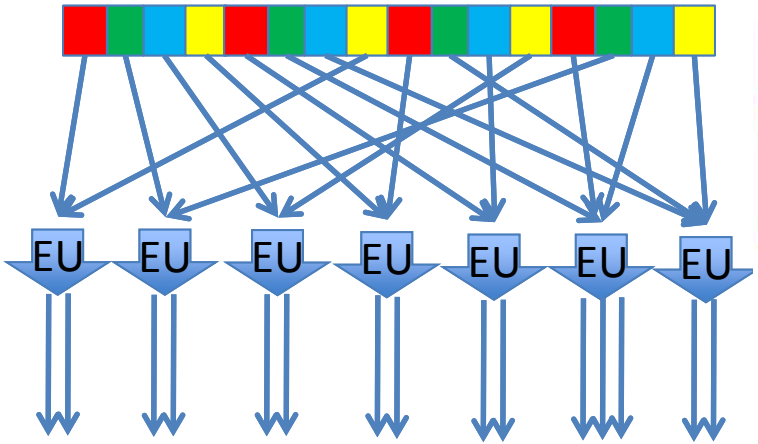
2. We seem to think we can read / modify / write the render target repeatedly…

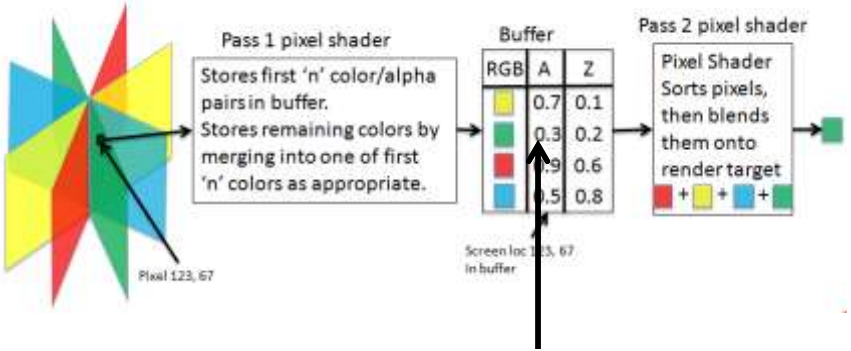
Spoiler alert: Algorithm will result in temporal artifacts – lets see why…

Our input comes from an array of EUs

Input pixels

EU's (Execution Units) going "flat out"!





No guaranteed order for UAV access events.

UAV access occurs in random order, so triplets after the first 'n' will be merged into different slots, so we don't get the same image every frame.

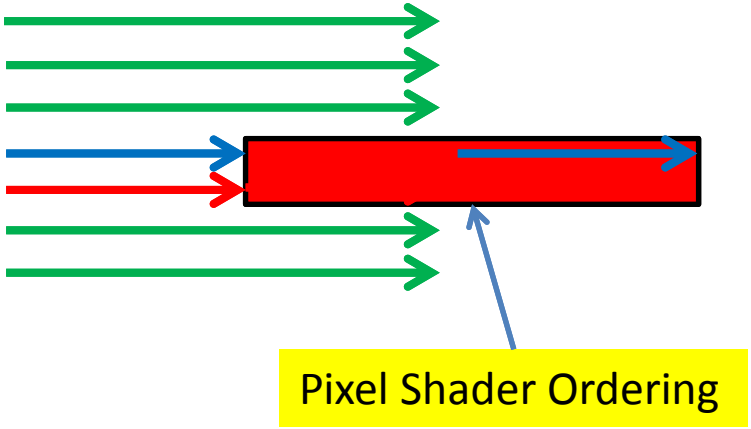
Net result: Nice idea then, but it won't work as-is…

Enter Pixel Shader Ordering to save the day!

Green threads writing to unique pixels
- Executed as normal for UAV
- Pipeline unchanged, all executed is in parallel.

Blue and red threads writing to same pixel
- In this case Red thread issued before blue
- Original render order reapplied
- Determinacy guaranteed.

Pixel Shader execution on multiple EUs



Only specific colliding threads are serialized to minimize pipeline interference.

How do you order a UAV?

Some new HLSL functions.

IntelExt_Init()

- Initializes the interface to the extensions.

IntelExt_BeginPixelShaderOrdering();

- All shader access to any UAV after the call will be serialized in original primitive order

How to Composite Transparent and Opaque Geometry

Traditional blend math using Alpha and (1 – Alpha)…

R*(1-a1)+c1*a1

(R*(1-a1)+c1*a1)*(1-a2)+c2*a2

((R*(1-a1)+c1*a1)*(1-a2)+c2*a2)*(1-a3)+c3*a3

Which we can reshuffle to give us this:

Alpha = (1-a1)*(1-a2)*(1-a3)

Color = c1*a1*(1-a2)*(1-a3)+c2*a2*(1-a3)+c3*a3

Which we can blend like this:

SrcBlend will be D3D11_BLEND_ONE

DstBlend will be D3D11_BLEND_SRC_ALPHA

BlendOp will be D3D11_BLEND_OP_ADD

Allowing us to blend our multiple Color + Alpha pairs to the render target In a single blend op.

About, Blend modes.

From DX Specifications:

In reality, we have 3 parameters to the blend pipeline:
SrcBlend – What to scale the value in the shader by
DstBlend – what to scale the value in the render target by
BlendOp – how to combine the 2 results

The important thing here is that this is all color blending.
If you want to get more complex, you cant…

Deferred rendering 101…

Geometry Buffer (G-Buffer):

- All scene models rendered to an intermediate buffer (called a G-Buffer)
- G-Buffer pixels contain normals, materials, texture information etc. in screen space.
- A second pass “composites” all this data with light data and shadow data to the screen.
- Well documented and well used.

Unanswered deferred rendering problem:

-How do I add a decal of, say, a bullet hole onto the G-Buffer?

Using Normal Decals on deferred surfaces

G-buffer
Normal data

To apply a bullet hole or an axe mark…
simply
- Render your G-Buffer
- Take a normal map of a bullet hole
- Blend it with the G-Buffer
- Result will be a correctly mapped bullet hole

But you can't do that with the blend stage in D3D!

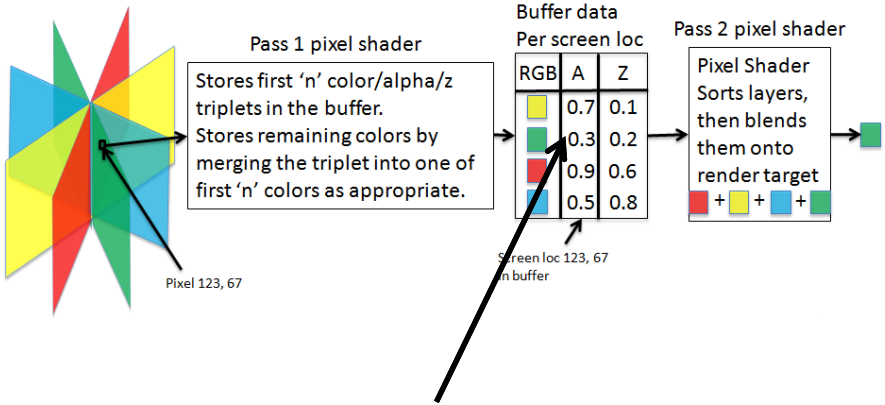
Blending Normals – Solved!

You can do it with a structured buffer and a UAV, if you use Pixel Shader Ordering to avoid clashes.

Normals can be safely blended together creating Correct surface deformations.

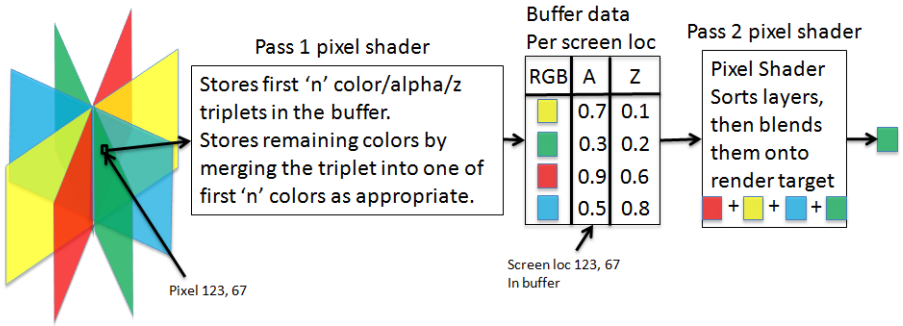
Potentially, this solves it completely!

Time to think…



The entries in the structured buffer do not need to be colors.
We could store normals there, or anything else for that matter!

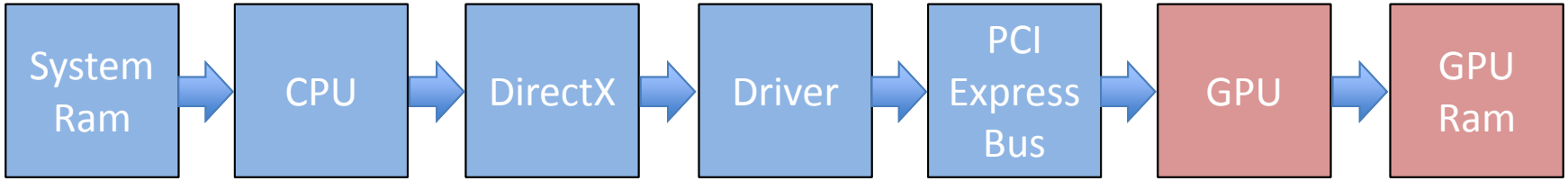
So far, then…



- How do I sort alpha polygons quickly? - DONE
- How do I sort alpha polygons correctly? - DONE
- How do I achieve programmable blending in DirectX? - DONE
- How do I transfer data to and from GPU memory efficiently?

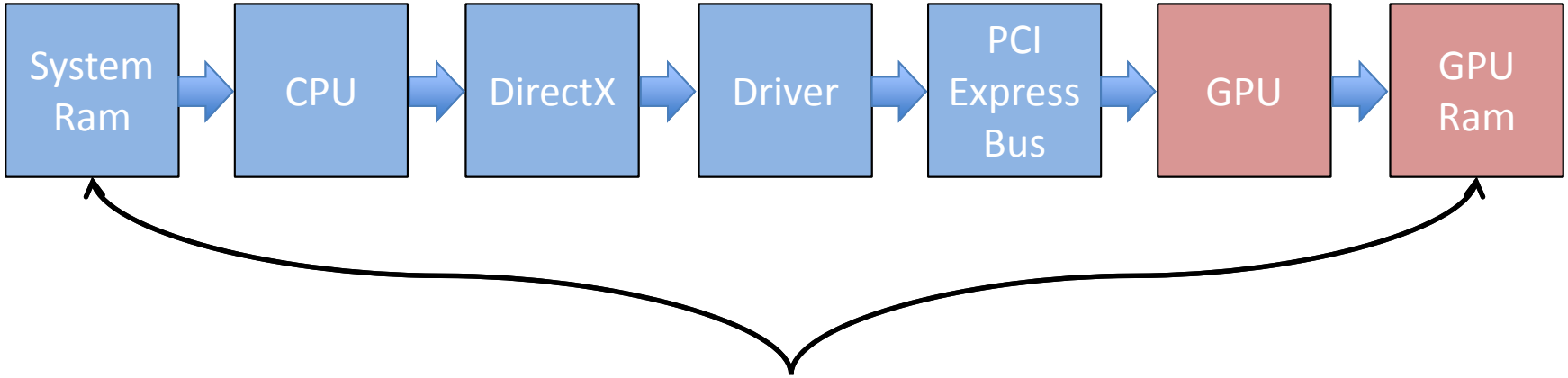
What do we mean, transfer data quickly?

To upload a texture, buffer, anything else…



Dictates load times
Restricts per frame upload & download volumes
Physically limits algorithm development

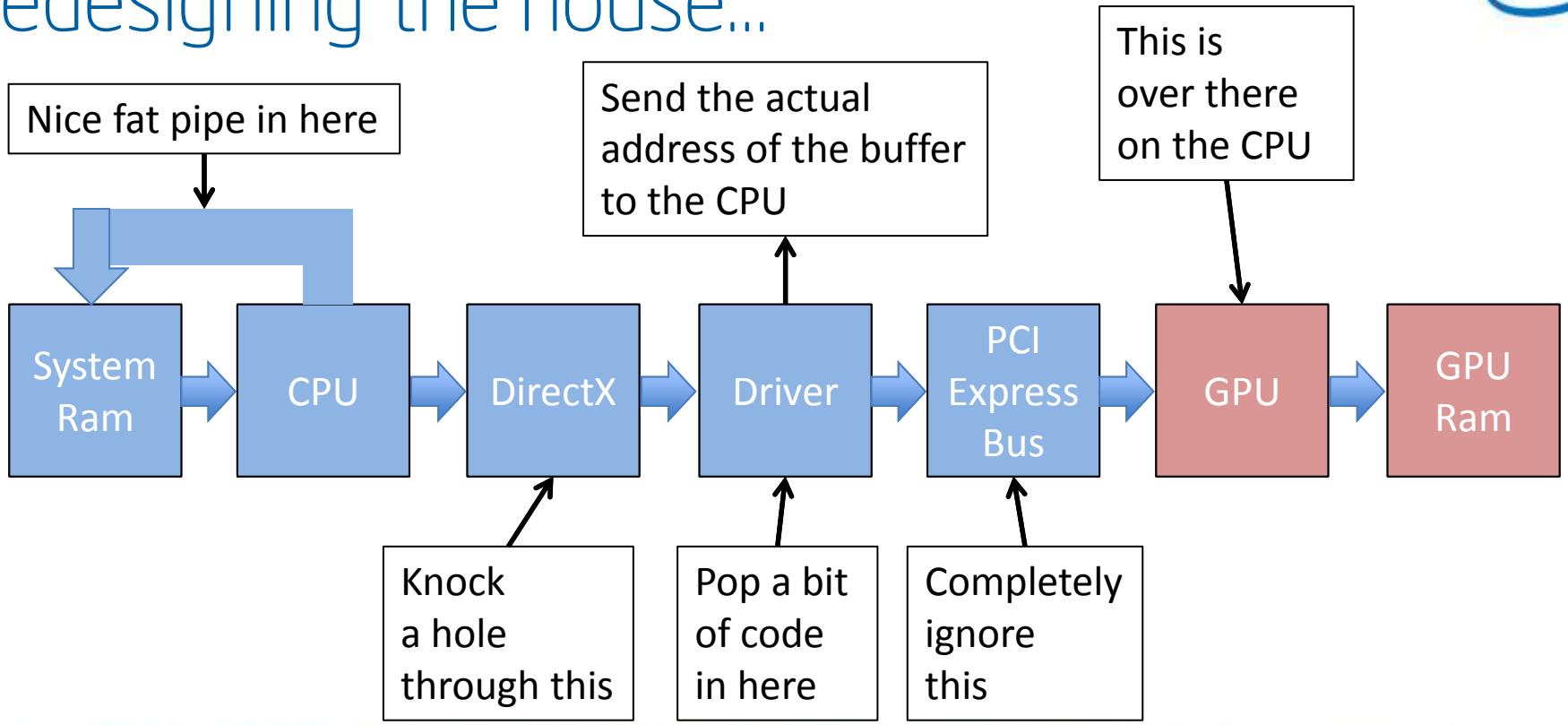
Realisation…



Wait a minute…
On processor graphics systems, these are the same thing…

Hmm, try telling DirectX that!

Redesigning the house...



The Instant Access extension.

```
CreateSharedTexture2D(
        LPD3D11Device ,
        D3D11_TEXTURE2D_DESC,
         LPD3D11Texture2D *gputex,
         LPD3D11Texture2D *cputex)


CopyResource(cputex, gputex);
```

Helper function to create a shared texture.

gputex will be its name in all things GPU side,
    binding etc

cputex will be its name on CPU side
    map etc

CopyResource() command trapped by driver
    Links the textures together

Both refer to the same memory location

How do you detect the extensions?

Include the helper library, then…

```
CAPS_EXTENSION intelExtCaps;
ZeroMemory( &intelExtCaps, sizeof(CAPS_EXTENSION) );

if( S_OK != GetExtensionCaps( pd3dDevice, &intelExtCaps ) )
    return E_FAIL;

if( intelExtCaps.DriverVersion < EXTENSION_INTERFACE_VERSION_1_0 )
            return E_FAIL;

return S_OK;
```

Concluding…

We asked:

How do I sort alpha polygons quickly?

You can't get much quicker than not doing it, sure there is a little bit of extra work in the shader.

We asked:

How do I sort alpha polygons correctly?

We found out there was no solution to some issues, until we changed the game with Pixel Shader Ordering.

Then we asked:

How do I achieve programmable blending in DirectX?

And we looked at one trick involving Pixel Shader Ordering which will change the game.

Finally, we asked:

How do I transfer data to and from GPU memory efficiently?

We found out that the Instant Access extension lets us copy to and from GPU ram very quickly.

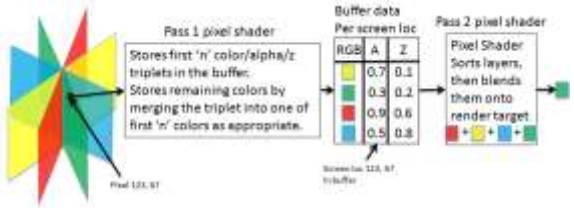
From Rome2: Total War. Showing complexity of modern transparency passes

And finally

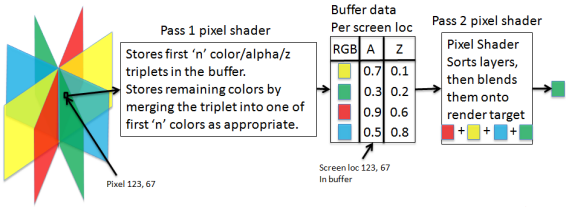
Always check out http://software.intel.com/sites/billboard/ to see what's new

Go check out the Intel Booth – some of this stuff is being demonstrated there. Lots of other cool stuff there too!

Don't forget to fill out the questionnaire.



Questions?


Pass 1 pixel shader
Stores first 'n' color/alpha/z triplets in the buffer.
Stores remaining colors by merging the triplet into one of first 'n' colors as appropriate.
Pixel 123, 67
Buffer data
Per screen loc
RGB
A
Z
0.7
0.1
0.3
0.2
0.9
0.6
0.5
0.8
Screen loc 123, 67
In buffer
Pass 2 pixel shader
Pixel Shader
Sorts layers, then blends them onto render target

The Next Session

- 2:30 p.m. to 3:30 p.m.
- **Cross-Platform Game Development: Best Practices Learned from Unreal, Unity, and More**
- Presenters: Omar Rodriguez and Orion Granatir

For more information: www.intel.com/software/gdc

Legal Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO SALE AND/OR USE OF INTEL PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.

Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Intel Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Intel may make changes to specifications, product descriptions, and plans at any time, without notice.

The Intel processor and/or chipset products referenced in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

All dates provided are subject to change without notice. All dates specified are target dates, are provided for planning purposes only and are subject to change.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

* Other names and brands may be claimed as the property of others.

Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel® Compiler User and Reference Guides" under "Compiler Options." Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101