

CS354R

DR SARAH ABRAHAM

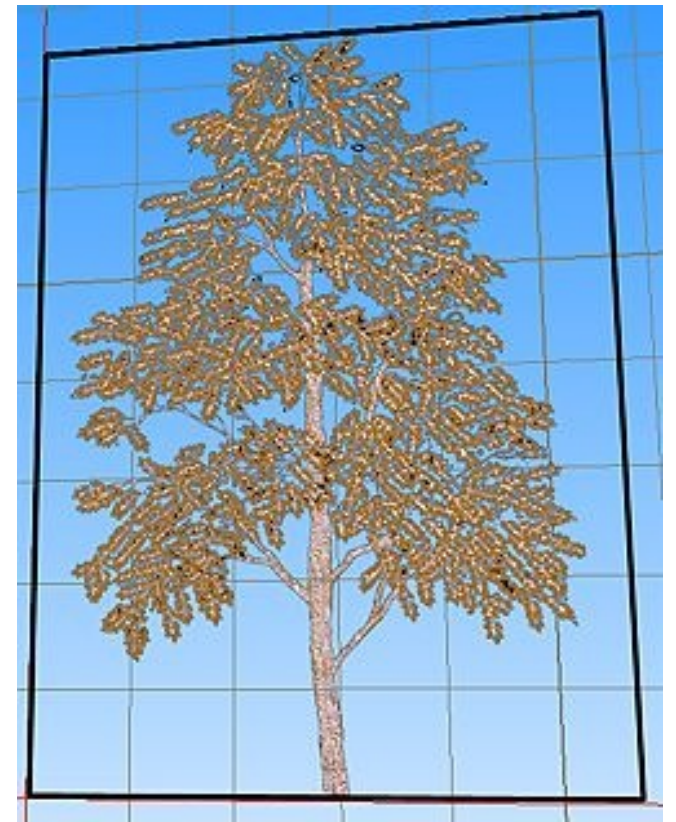
BILLBOARDS AND SPRITES

EXTREME LOD

- ▶ What can a mesh turn into at its most distant LOD?

BILLBOARDS

- ▶ A billboard is extreme Level of Detail (LOD), reducing all the geometry to one or more textured polygons
 - ▶ Also considered a form of image-based rendering
- ▶ Questions in designing billboards:
 - ▶ How are they generated?
 - ▶ How are they oriented?
- ▶ Also called sprites, but a sprite normally stays aligned parallel to the image plane



HOW TO GENERATE BILLBOARDS?

- ▶ By hand – a skilled artist does the work
 - ▶ Paints color and alpha
 - ▶ May generate a sequence of textures for animating
- ▶ Automatically:
 - ▶ Render a complex model and capture the image
 - ▶ Alpha detected by looking for background pixels in the image
 - ▶ Blend out alpha at the boundary for good anti-aliasing

HOW TO CONFIGURE BILLBOARDS?

- ▶ The billboard polygons can be laid out in different ways
 - ▶ Single rectangle
 - ▶ Two rectangles at right angles
 - ▶ Several rectangles about a common axis
 - ▶ Several rectangles stacked
- ▶ Issues are:
 - ▶ What sorts of billboards are good for what sorts of objects?
 - ▶ How is the billboard oriented with respect to the viewer?

SINGLE POLYGON BILLBOARDS

- ▶ The billboard consists of a single textured polygon
- ▶ It must be pointed at the viewer, or it would disappear when viewed from the side
- ▶ Point Sprites:
 - ▶ Billboard rotated about a central point that faces the camera
- ▶ Axis Billboards:
 - ▶ Billboard aligned along an axis (arbitrary or axis-aligned)

ALIGNING A BILLBOARD

- ▶ Billboard has a known forward vector F that points out from the face
- ▶ Billboard has an “up” or axis vector A
- ▶ Camera has a view direction V
- ▶ How can we realign F to face V ?

ALIGNMENT ABOUT AXIS

- ▶ **A** is billboard axis, **V** is viewer direction.
From current forward **F** move to
desired forward **D**
- ▶ Calculate **D**
- ▶ Compute angle γ between **F** and **D**
- ▶ Significant shortcut if **A** is the z axis,
and **F** points along the x axis

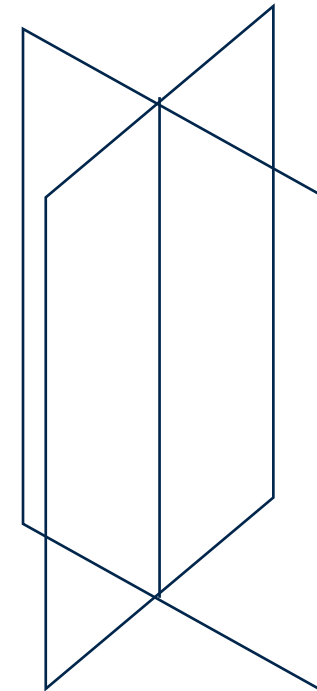
$$\mathbf{D} = \mathbf{A} \times (\mathbf{V} \times \mathbf{A})$$

$$\gamma = \cos^{-1} \left(\frac{\mathbf{F} \cdot \mathbf{D}}{\|\mathbf{F}\| \|\mathbf{D}\|} \right)$$

$$\gamma = \tan^{-1} \left(\frac{V_y}{V_x} \right)$$

MULTI-POLYGON BILLBOARDS

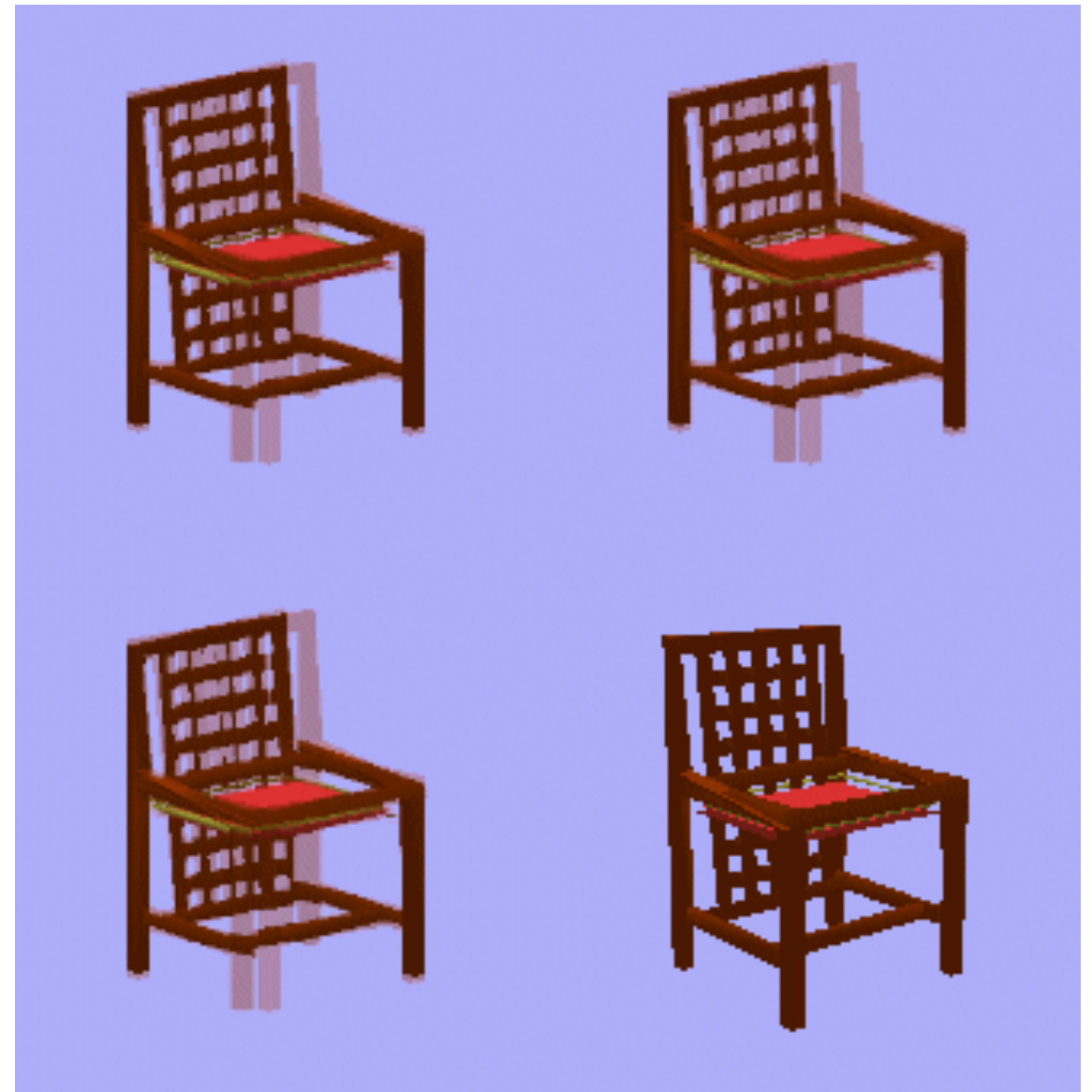
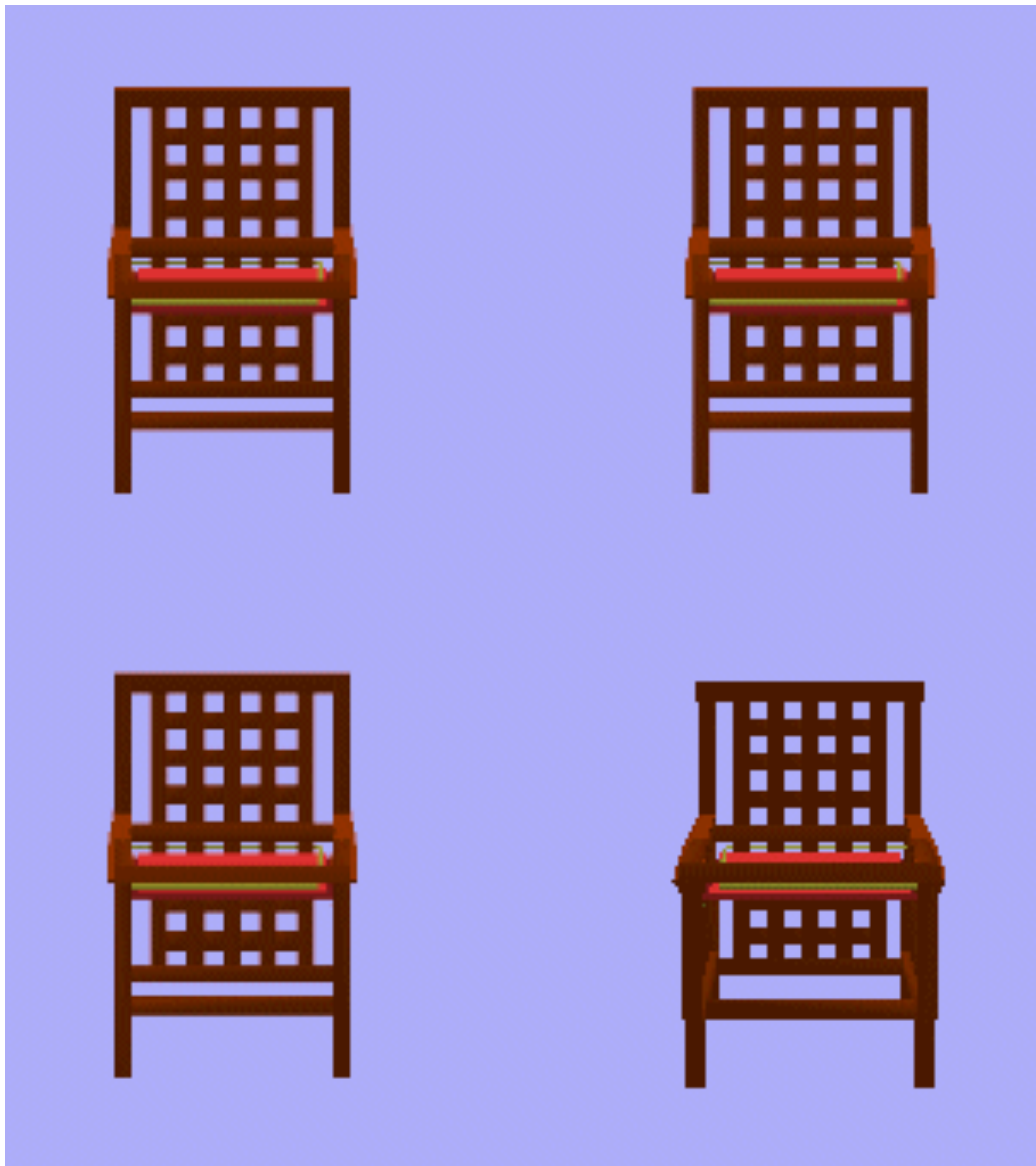
- ▶ Use two polygons at right angles:
 - ▶ No alignment with viewer
 - ▶ What is this good for?
- ▶ Use more polygons for better appearance
- ▶ Rendering options: Blended or just depth buffered



VIEW DEPENDENT BILLBOARDS

- ▶ What if the object is not rotationally symmetric?
 - ▶ Appearance should change from different viewing angles
- ▶ This can be done with billboards:
 - ▶ Compute multiple textures corresponding to different views
 - ▶ Keep polygon fixed but vary texture according to viewer direction
 - ▶ Interpolate with texture blending between the two nearest views
 - ▶ Use 3D textures and hardware texture filtering to achieve good results
- ▶ Polygons are typically fixed, restricting the viewing angles
 - ▶ Use more polygons that each have a set of views associated with it

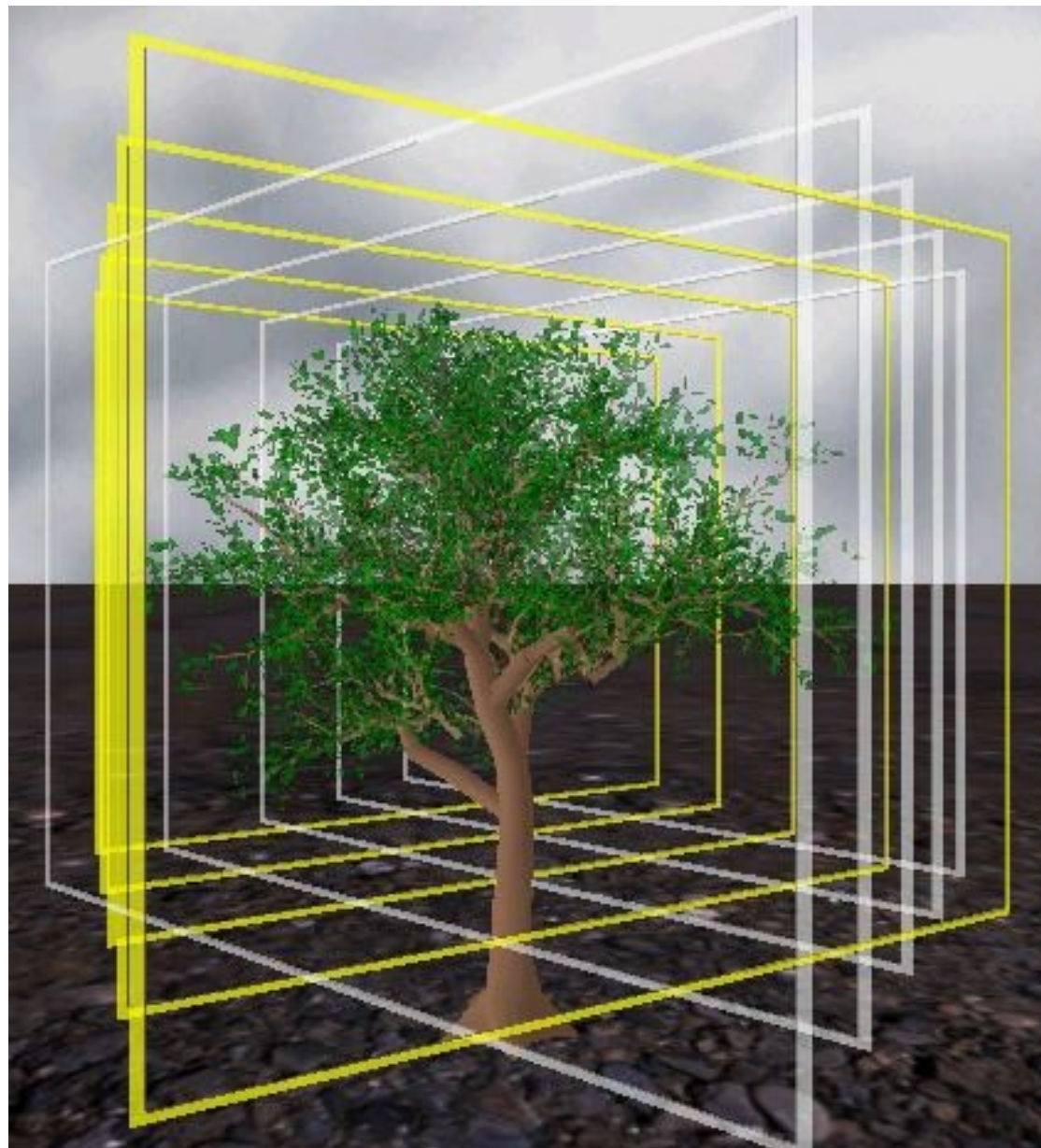
VIEW DEPENDENT BILLBOARDS



(Nvidia)

IMPOSTOR EXAMPLE

- ▶ Another method uses slices from the original volume and blends them



BILLBOARDING IN ACTION

- ▶ Fire in...pretty much any game
 - ▶ Tomb Raider <<https://youtu.be/U-Lx9luwwXc?t=218>>
- ▶ You can probably also catch some of the billboarding used for low-res LODs if you pay attention to objects in the distance
 - ▶ But particle effects are very frequently billboarded even at high resolutions

ADDITIONAL OPTIMIZATIONS

- ▶ How do we optimize geometry in scenes besides using LODs?

PIPELINE EFFICIENCY

- ▶ The rendering pipeline is (as the name suggests) a pipeline
 - ▶ Slowest pipeline operation determines throughput (frame rate)
 - ▶ For graphics, that could be memory bandwidth, transformations, clipping, rasterization, lighting, buffer fill etc
- ▶ Profiling tools tell you which part of your pipeline is slow
- ▶ One speedup is reducing the complexity of the geometry
 - ▶ Impacts every part of the pipeline up to the fragment stage
 - ▶ Assumption: You will touch roughly the same pixels, even with simpler geometry

REDUCING GEOMETRY

- ▶ Assume we are living in a polygon mesh world
- ▶ Several strategies exist, with varying degrees of difficulty, reductions in complexity, and quality trade-offs:
 - ▶ Reduce the amount of data sent per triangle, but keep the number of triangles the same
 - ▶ Reduce the number of triangles by ignoring things that you know the viewer can't see – *visibility culling*
 - ▶ Reduce the number of triangles in view by reducing the quality (maybe) of the models – *level of detail (LOD)*

COMPRESSING MESHES

- ▶ Base case: Three vertices per triangle with full vertex data (color, texture, normal etc)
- ▶ Much of this data is redundant:
 - ▶ Triangles share vertices
 - ▶ Vertices share colors and normals
 - ▶ Vertex data may be highly correlated
- ▶ Compression strategies seek to avoid sending redundant data
- ▶ Impacts memory bandwidth, but not too much else
 - ▶ A concern for transmitting models over a network

COMPRESSION OVERVIEW

- ▶ Use triangle strips to avoid sending vertex data more than once
- ▶ Use vertex arrays
 - ▶ Tell the API what vertices will be used
 - ▶ Specify triangles by indexing into the array
 - ▶ Reduces cost per vertex
 - ▶ Allows hardware to cache vertices
- ▶ Non-shared attributes, such as normal vectors, limit the effectiveness of some of these techniques
- ▶ These techniques are required in OpenGL ES but good practice even when not space/operation restricted

MESH COMPRESSION

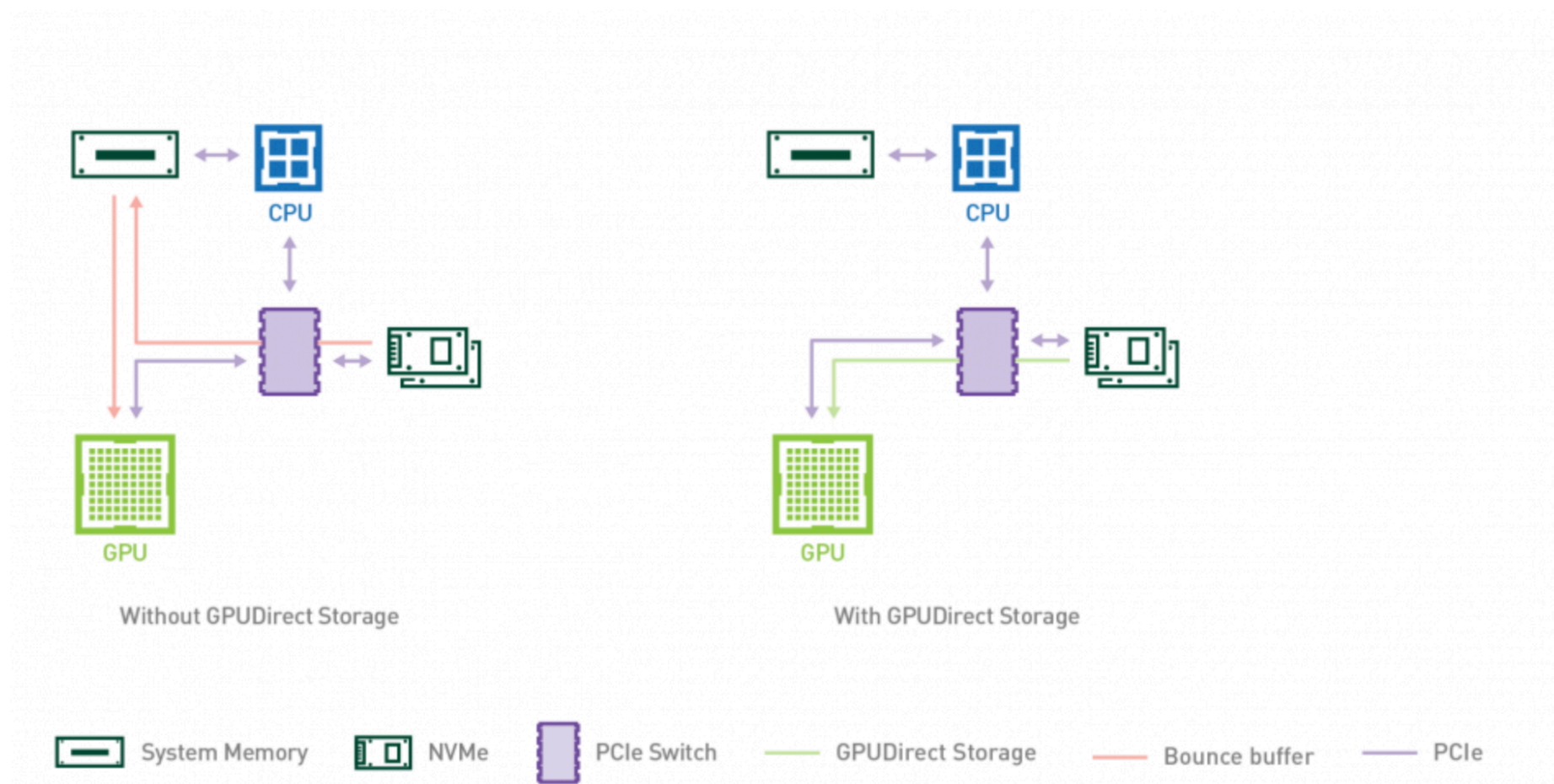
- ▶ Pipelined hardware typically accepts data in a stream, and has small buffers
 - ▶ Can't do decompression that requires holding the entire mesh, or any large data structure
- ▶ Do decompression in software
- ▶ Typical strategies
 - ▶ Treat connectivity (which vertices go with which triangles) separately from vertex attributes (location, normal etc)
 - ▶ Build long strips or other implicit connectivity structures
 - ▶ Apply standard compression techniques to vertex attributes

GPU DIRECT MEMORY ACCESS

- ▶ DMA (Direct Memory Access) allows for asynchronous memory access independent of the CPU
 - ▶ Useful for large data transfers and during I/O
- ▶ GPUs have high latency
 - ▶ Modern tasks (e.g. big data, AI, etc) require massive data sets resulting in I/O bottleneck
- ▶ Use of GPU-specific DMA allows for faster access across bus or over a network

NVIDIA GPUDIRECT STORAGE

- ▶ Allows for direct access by GPU
- ▶ Avoids overhead of bounce buffer (typically required to process I/O from host to subsystem)



PERFORMANCE TAKE AWAYS

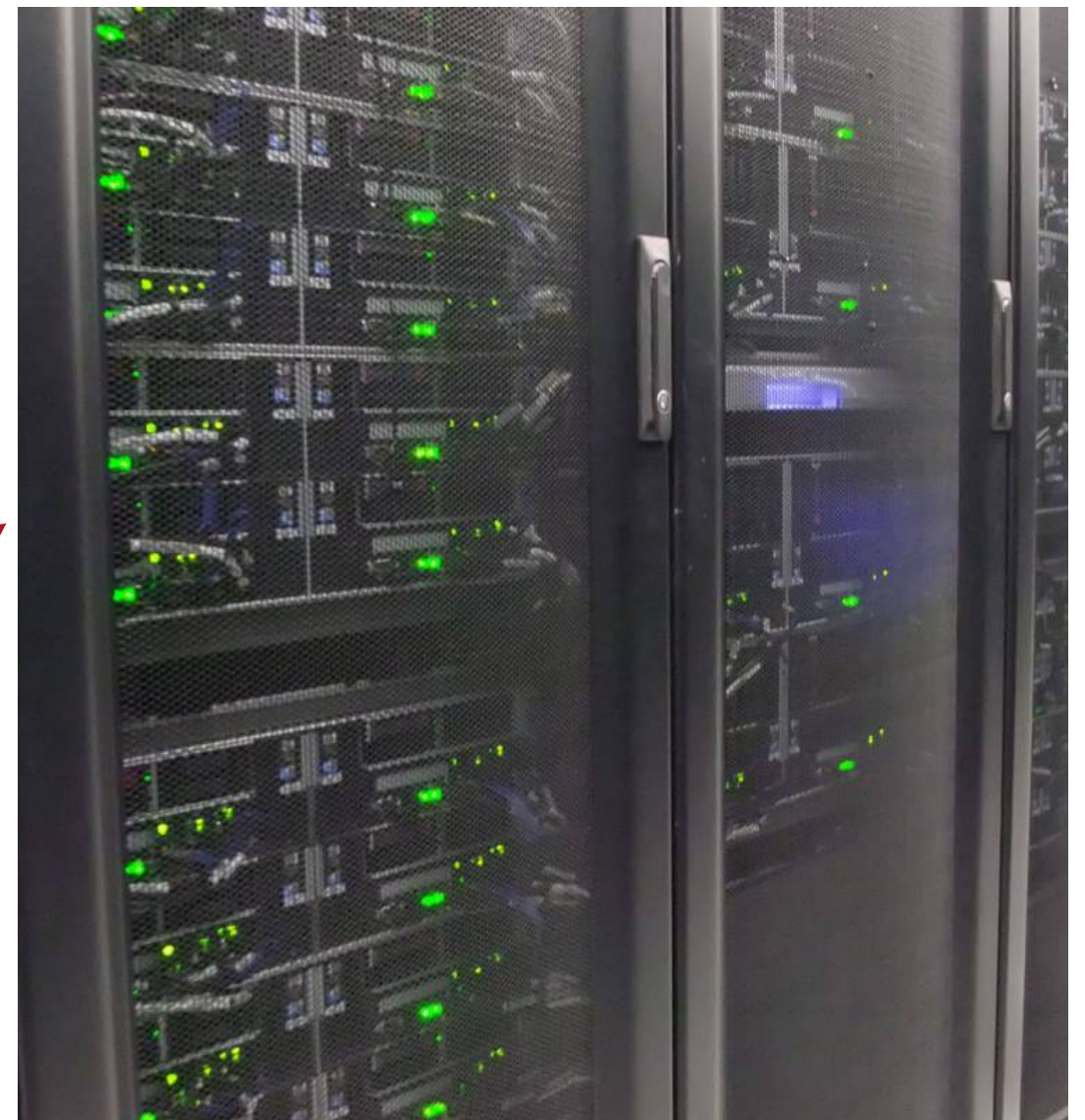
- ▶ Hardware affects how we interact with software
 - ▶ Cannot fully abstract away low-level considerations for performant code
- ▶ System needs are changing as big data gets even bigger and more prevalent
 - ▶ I/O has always been slow but now it's too slow...
- ▶ Old techniques are not irrelevant!
 - ▶ The use case may be different but the principles are the same

CONSIDERING CLOUD GAMING...

- ▶ Concept of game streaming services
 - ▶ Game processed remotely
 - ▶ Results sent to player's device



data sent
fun returned



Wow! Games are sure fun!

CLOUD GAMING

- ▶ What are our considerations?
 - ▶ What should be processed remotely versus on device?
 - ▶ What should the packets of data contain?
 - ▶ What about AR/VR?

BILLBOARDING HOW-TOS

- ▶ NeHe Productions <http://nehe.gamedev.net/article/billboarding_how_to/18011/>
- ▶ Lighthouse 3D <<http://www.lighthouse3d.com/opengl/billboarding/>>
- ▶ NVidia GPUDirect Storage <<https://developer.nvidia.com/blog/gpudirect-storage/>>