CS354R

DR SARAH ABRAHAM

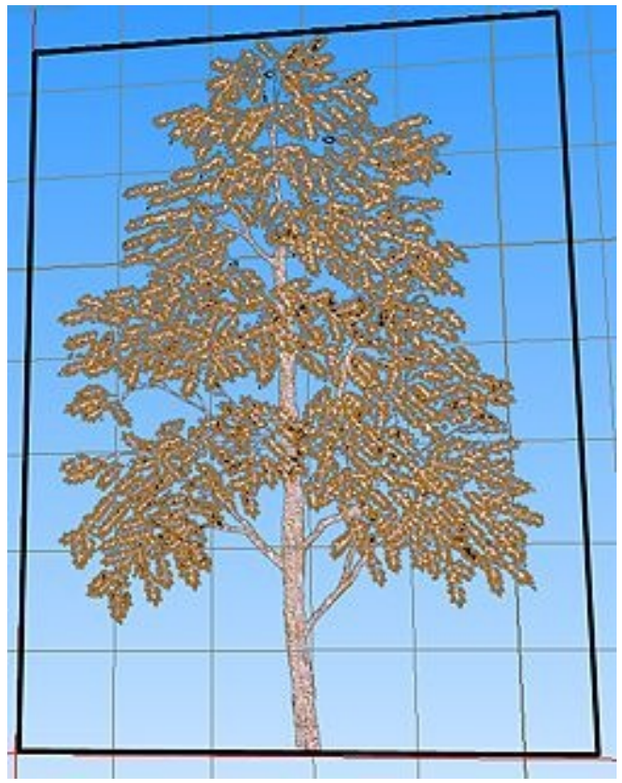
FAST SHADING

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**

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

- Compute angle γ between **F** and **D**

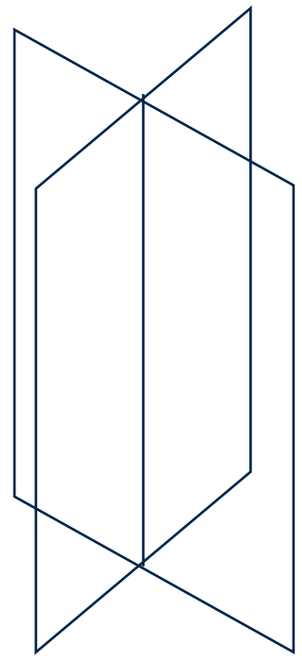
$$\gamma = \cos^{-1}\left(\frac{\boldsymbol{F} \bullet \boldsymbol{D}}{\|\boldsymbol{F}\|\|\boldsymbol{D}\|}\right)$$

- Significant shortcut if **A** is the z axis,
  and **F** points along the x axis

$$\gamma = \tan^{-1}\left(\frac{\boldsymbol{V}_y}{\boldsymbol{V}_x}\right)$$

MULTI-POLYGON BILLBOARDS

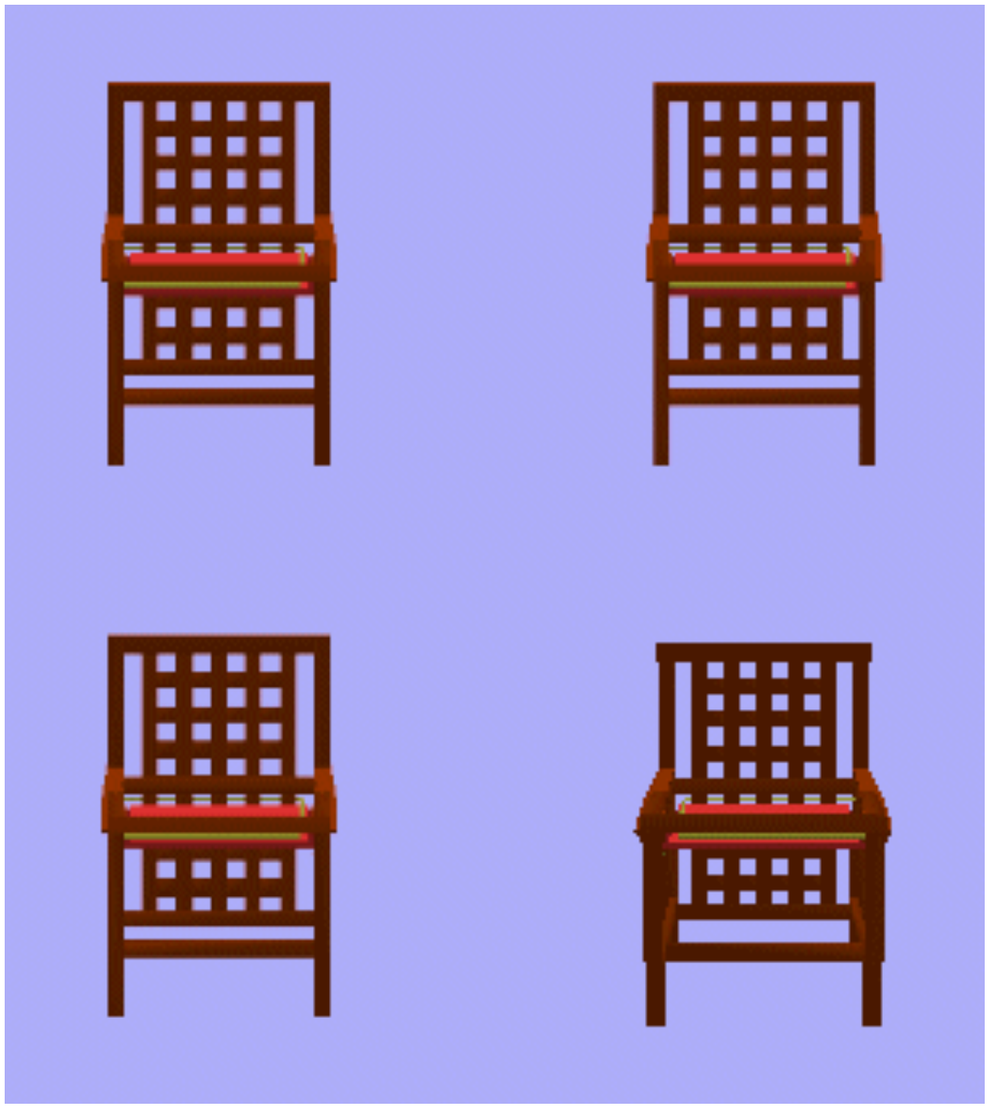
- Use two polygons at right angles:
  - No alignment with viewer
  - What is this good for?
- Use more polygons for better appearance
- Rendering options: Blended or just depth buffered

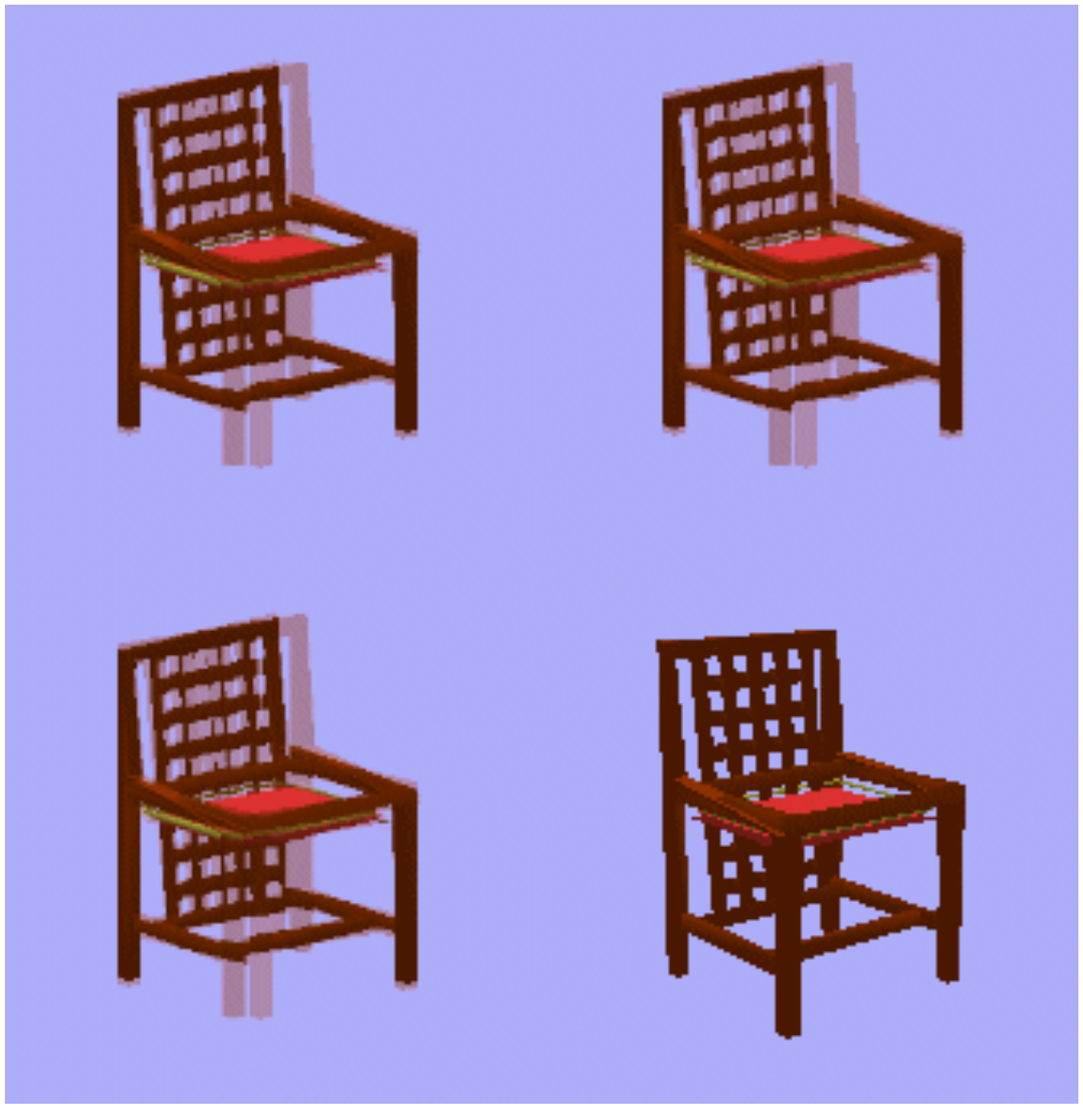


VIEW DEPENDENT BILLBOARDS

- 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

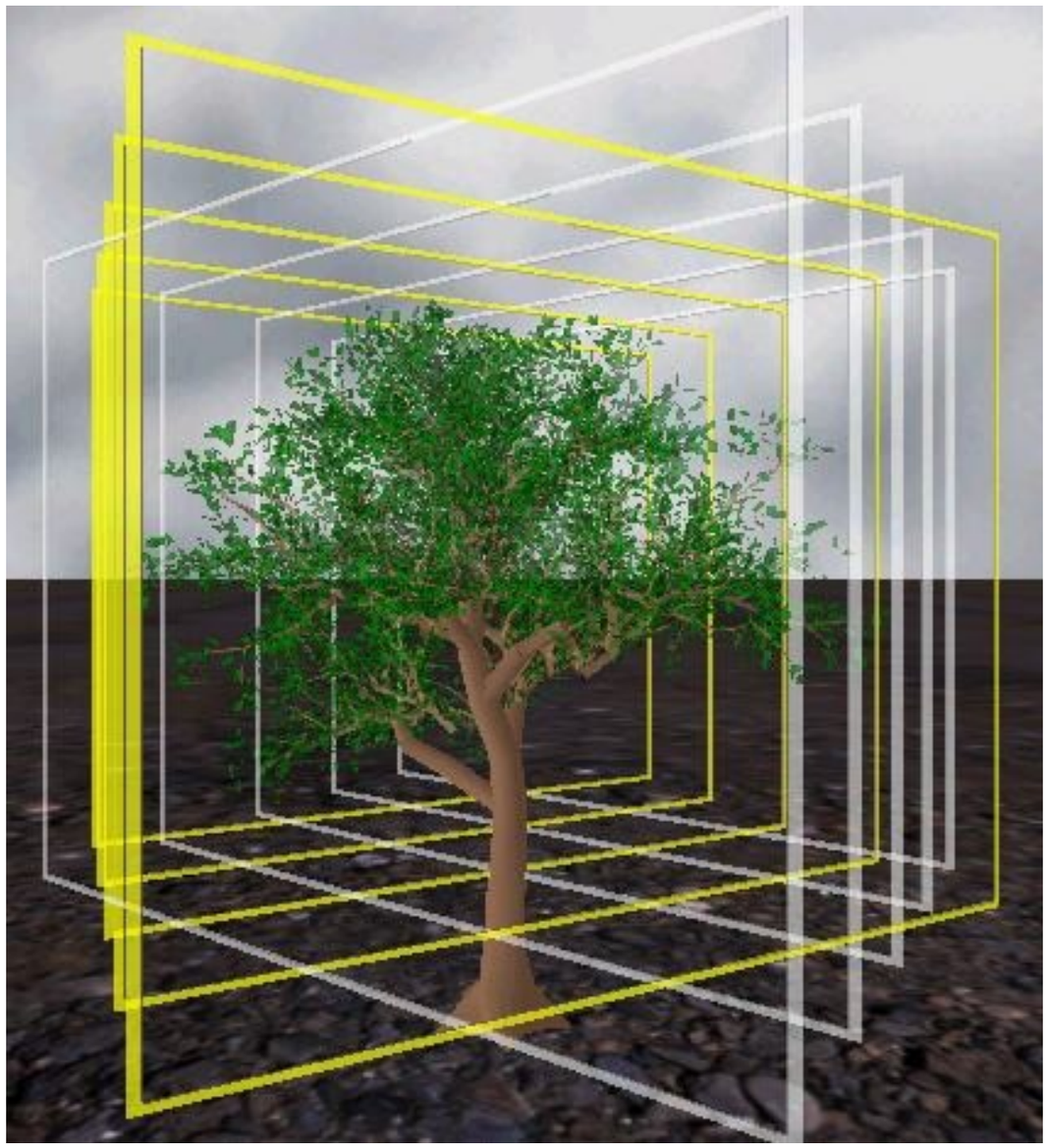




(Nvidia)

IMPOSTOR EXAMPLE

- Another methods uses slices from the original volume and blends them



BILLBOARDING IN ACTION

- Fire in…pretty much any game
  - Tomb Raider <https://youtu.be/U-Lx9IuwwXc?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?

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

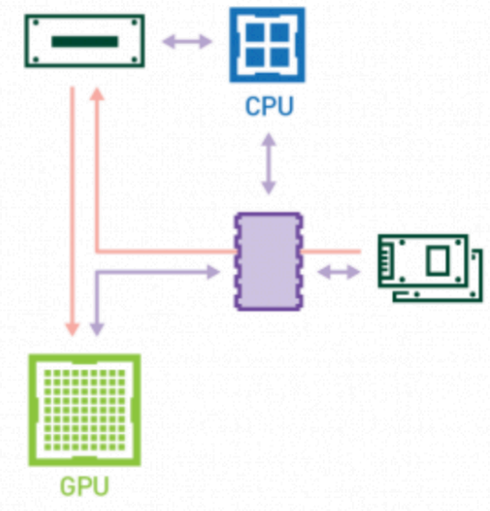
- Use triangle strips to avoid sending vertex data more than once
- Use vertex arrays
  - Tell the API what vertices will be used
  - Specify triangles by indexing into the array
  - Reduces cost per vertex
  - Allows hardware to cache vertices
- Non-shared attributes, such as normal vectors, limit the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

NOTE ON HARDWARE: GPU DIRECT MEMORY ACCESS

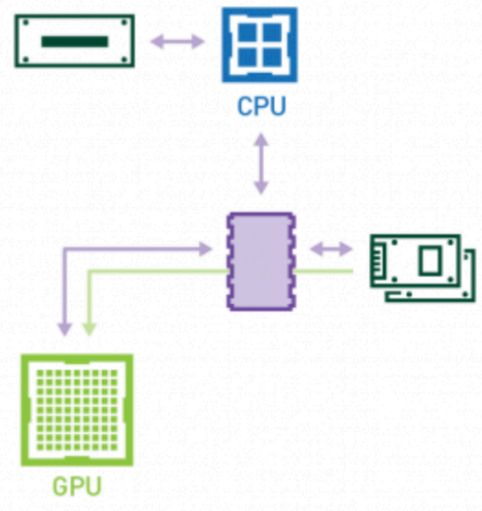
- DMA (Direct Memory Access) allows for asynchronous memory access independent of the CPU
  - Useful for large data transfers and during I/O
- GPUs have high latency
  - Modern tasks (e.g. big data, AI, etc) require massive data sets resulting in I/O bottleneck
- Use of GPU-specific DMA allows for faster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



Without GPUDirect Storage

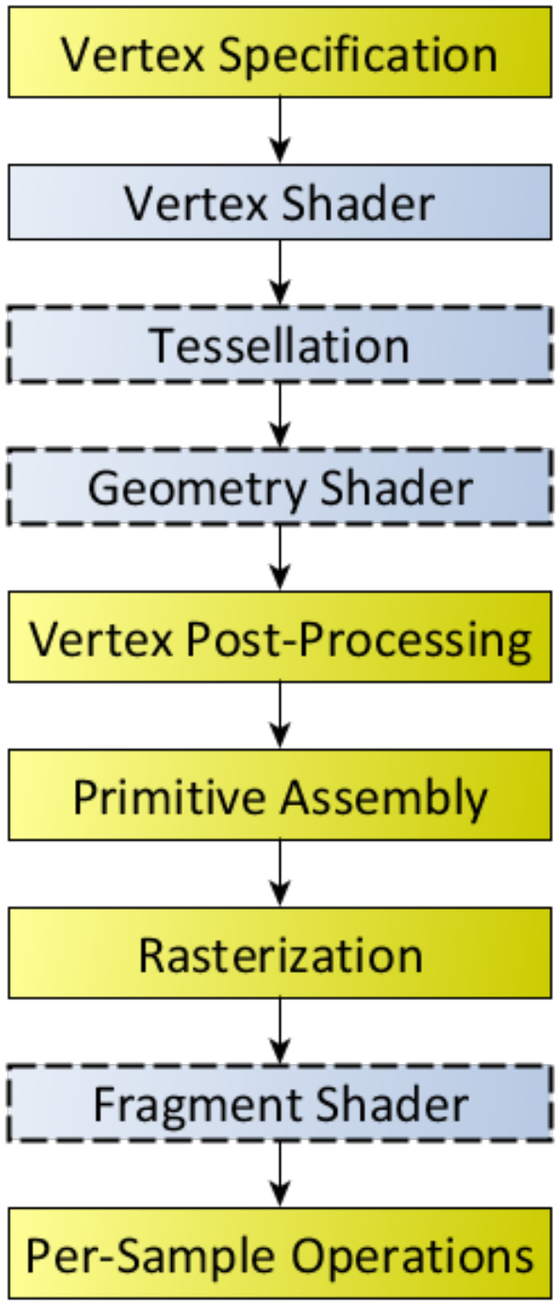


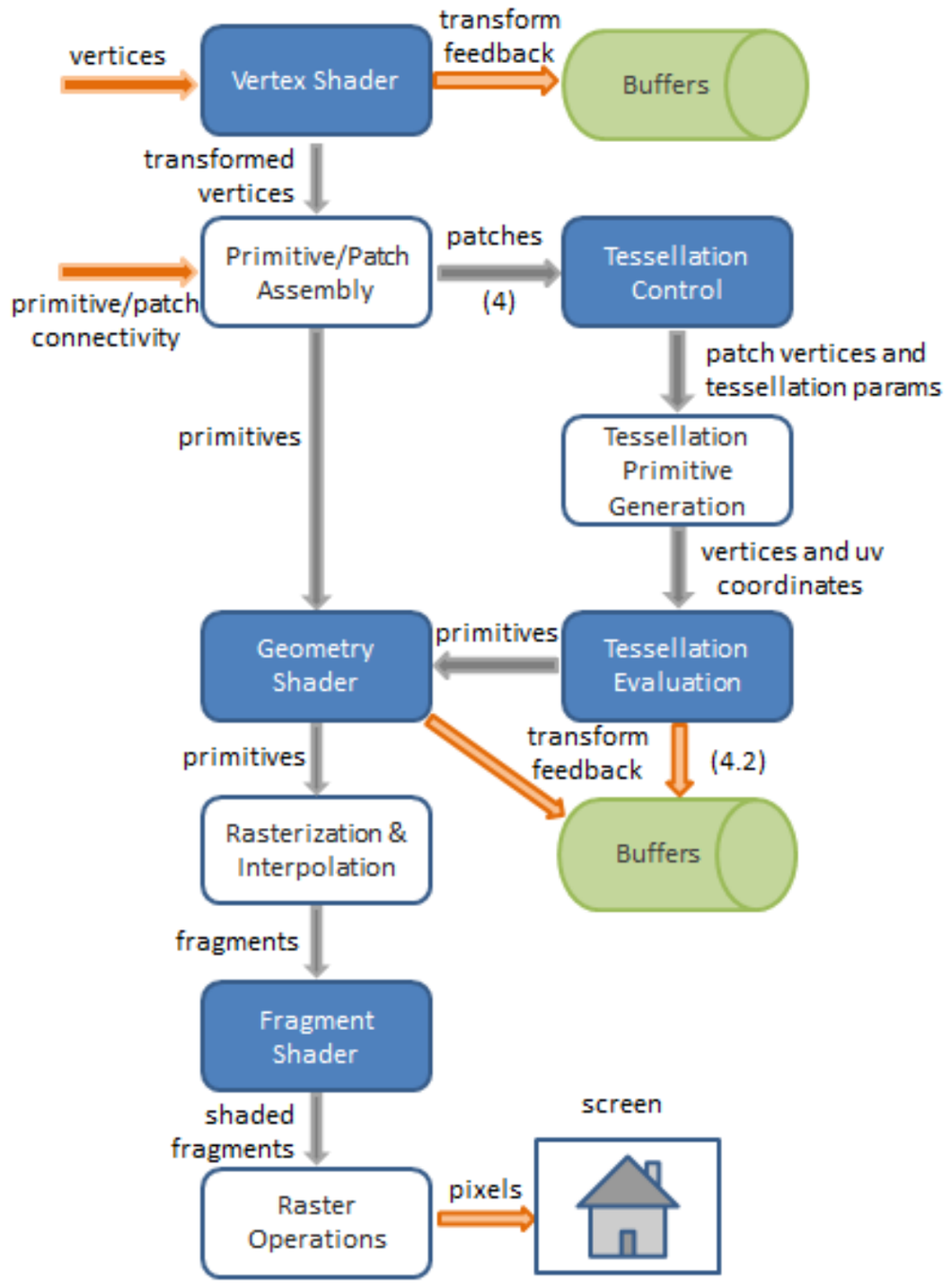
With GPUDirect Storage

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 can tell you which part of the pipeline is slow

RASTER PIPELINE





FORWARD SHADING

- Forward shading assumes shaders process everything in a serial fashion:
  - Process all scene vertices
  - Create necessary primitives
  - Rasterize primitives to screen based on depth
  - Color pixels based on fragment color
- Performance issues with increased lighting complexity
  - Objects processed wether or not they’re visible to the viewer

DEFERRED SHADING PIPELINE

- Defers expensive light calculations till scene complexity is reduced
- Scene geometry treated as textures within the fragment shader
- Only need to consider scene in per-pixel way
- Can better manage light complexity
- Can be combined with forward rendering and post-processing techniques

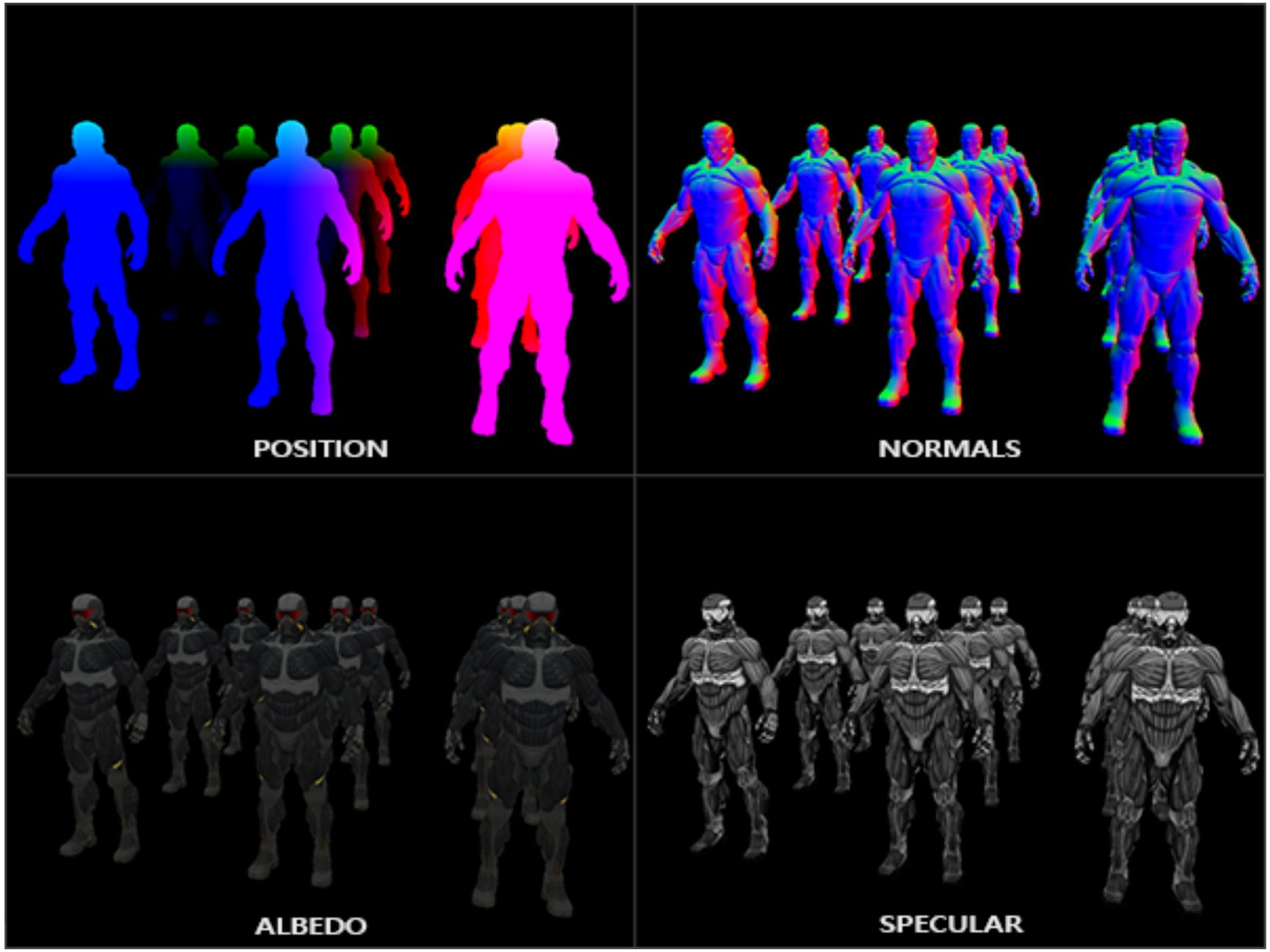
DEFERRED SHADING PASSES

- Rasterization broken into two passes:
  - Geometry pass
  - Lighting pass
- Geometry pass stores geometric information into **G-buffer**
- Lighting pass uses data in G-buffer to reconstruct scene but calculates lights per-pixel

CREATING THE G-BUFFER

- Contains textures that hold world-space data needed for final lighting pass
- Depth buffer has already determined what information is needed per pixel and culled all data that's not relevant
- Flexible texture precision allows for compact storage of the data

G-BUFFER DATA EXAMPLE



RUNNING A LIGHTING PASS

- Lighting applied to G-buffer content rather than the scene
  - One lighting operation per pixel
- Optimizations using light volumes
  - Allows for fast light attenuation

MODERN TECHNIQUES IN UNREAL

- **Nanite** handles culling and LODs on the GPU using compute shaders
- **Lumen** handles global illumination in a raster-driven way
  - Multiple distance fields and a hierarchical Z-buffer
- **MegaLights** allows for extreme number of dynamic and and shadowed area lights
  - Uses importance sampling for shadows and volumetric fog
  - Replaces BRDF and light evaluation in deferred shading

DEMOS

- Nanite and Lumen Demo:
  - https://www.youtube.com/watch?v=qC5KtatMcUw
- MegaLights Demo:
  - https://www.youtube.com/watch?v=hzcsKvrF-Ho

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/>