Light-field Display Architecture and the Complexities of Light-field Rendering

FoVI 3D

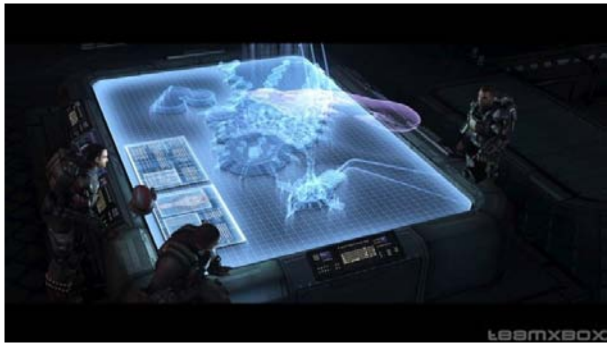
- FoVI 3D has a proven light-field display architecture for large format interactive light tables
  - Developed with support from the Department of Defense
    - DARPA
    - AFRL
    - ARL
    - Navy
  - No glasses, head tracking, or “tricks” required for naturally viewing 3D images
- Spin out from Zebra Imaging whose sole purpose is the development of dynamic light-field displays
- Multiple interactive light-field prototypes being utilized in DoD laboratories to derive requirements, understand the human machine interface, and determine the efficacy of natural 3D visualization

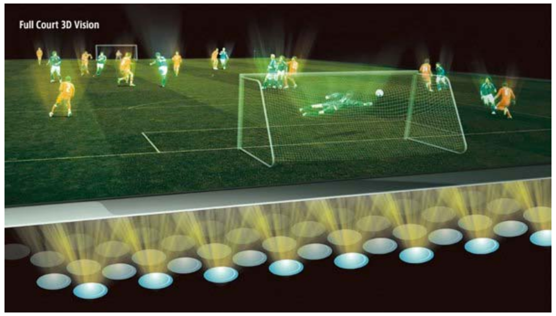
Thomas Burnett

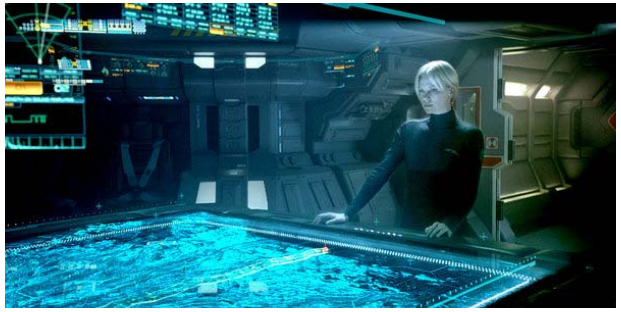
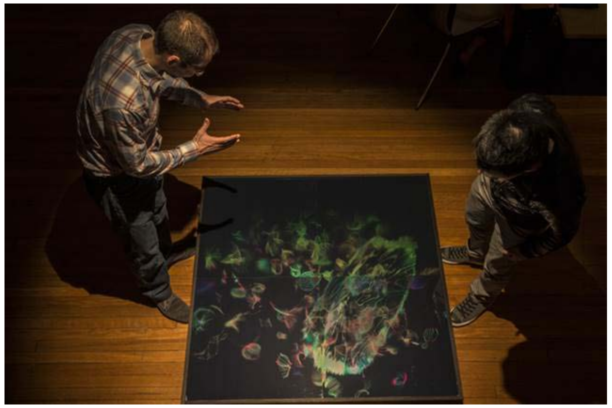
A founder and primary investigator for FoVI3D. ~15 years experience developing rendering solutions and architectures for static and dynamic light-field display systems.

tburnett@fovi3D.com

Light-field Display in the Movies and Media



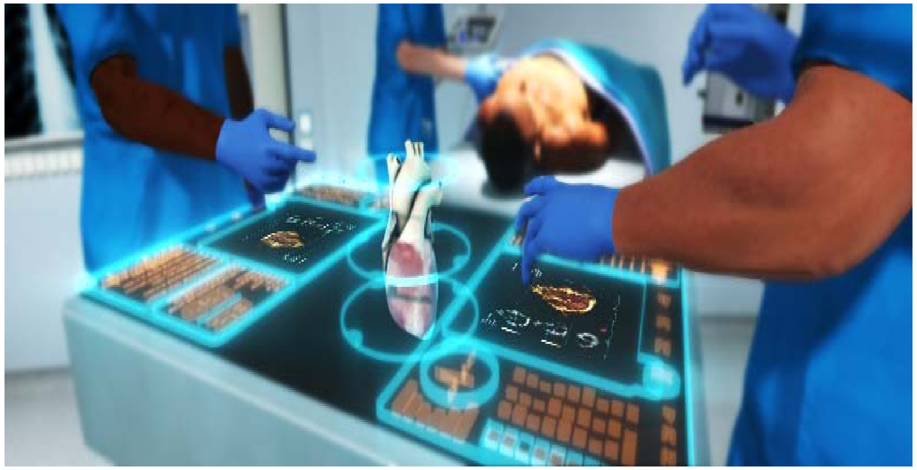


FoVI3D

Significance of Light-field Displays

- Human binocular vision and acuity and the accompanying 3D retinal processing of the human eye and brain are specifically designed to promote situational awareness and understanding in the natural 3D world.
- The ability to resolve depth within a scene, whether natural or artificial, improves our spatial understanding of the environment and as a result reduces the cognitive load accompanying the analysis and collaboration on complex tasks.
- A light-field display projects 3D imagery that is visible to the unaided eye (without glasses or head tracking) and allows for perspective correct visualization within the display's projection volume.
- Binocular disparity, occlusion, specular highlights and gradient shading, and other expected depth cues are correct from the viewer's perspective as in the natural real-world light-field.

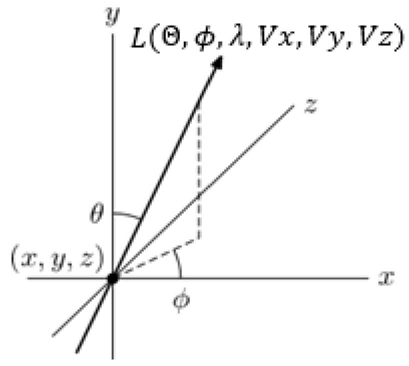


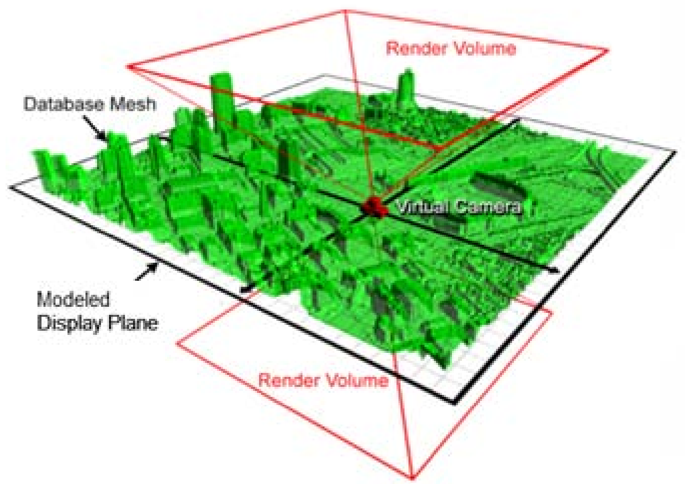
Light-field Radiance Image

A light-field can be described as a set of rays that pass through every point in space and is typically parameterized for computer vision in terms of a plenoptic function:

$$L = P(\Theta, \phi, \lambda, Vx, Vy, Vz)$$

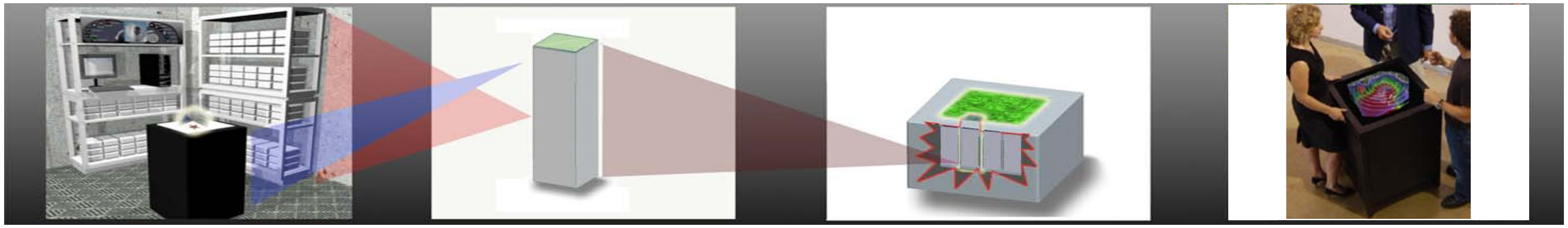
Where (Vx, Vy, Vz) defines a point in space, (Θ, ϕ) defines the orientation or direction of a ray, and λ defines the wavelength. The light-field display radiance image is a raster description of a light-field which can be projected through a microlens array to reconstruct a 3D image in space. The radiance image consists of a 2D array of hogels (holographic element) where the pixels represent the origin, direction, and intensity of light rays within the light-field as described by the plenoptic function.





A hogel is a radiance image for a single micro-lens. Light-field displays require many hogels and thus extremely large radiance images.

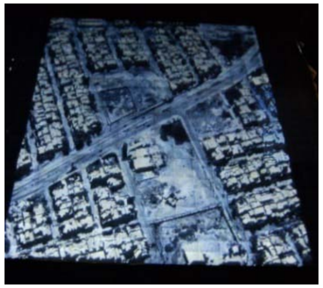
DARPA Urban Photonic Sand-table Display (UPSD) Program



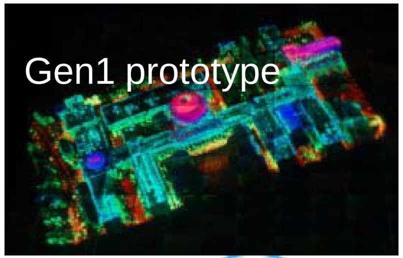
Four-phase program to develop dynamic 3D holographic displays

- ✓ Phase 1: Demonstration prototype – 2006 Apr
  - Developed initial technologies for dynamic 3D
  - Designed and built technology “Demonstrator” POC display
    - One-sq.ft. monochrome display; 10-mm resolution

- ✓ Phase 2: Advanced technology – 2007 July
  - 7x quality improvement
  - 11x component cost reduction
  - Scalable modularity
- ✓ DARPA Tech 2007 Conference:
  - Successful technology demonstration

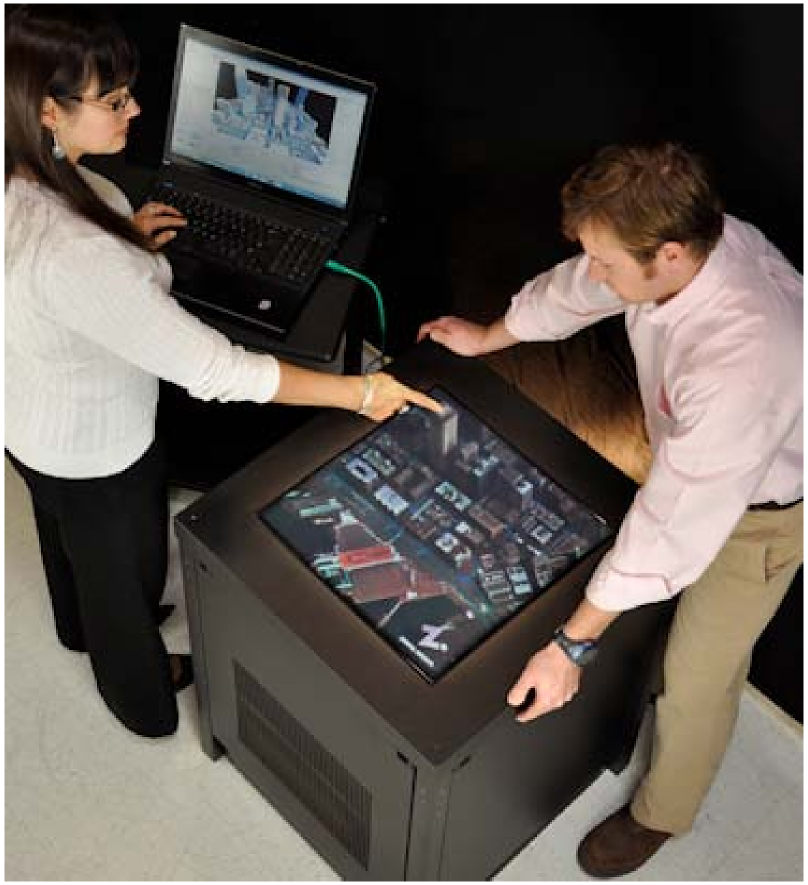


- ✓ Phase 3: Large non-color alpha prototype – 2008 June
  - Prototype: ~1 meter diagonal with better image quality
  - System consolidation & cost reductions



- ✓ Phase 4: Color alpha prototype – 2010 Dec
  - Prototype: improved image quality & color
  - Reduce cost: aggressive target for production cost (<$30/cm^2)
  - Increase manufacturability, yield & reliability
  - Establish low-cost assembly/QA systems and processes
  - Develop sophisticated operating software to support integration with initial customers
  - Initial applications/integration driven by transition partners and customers

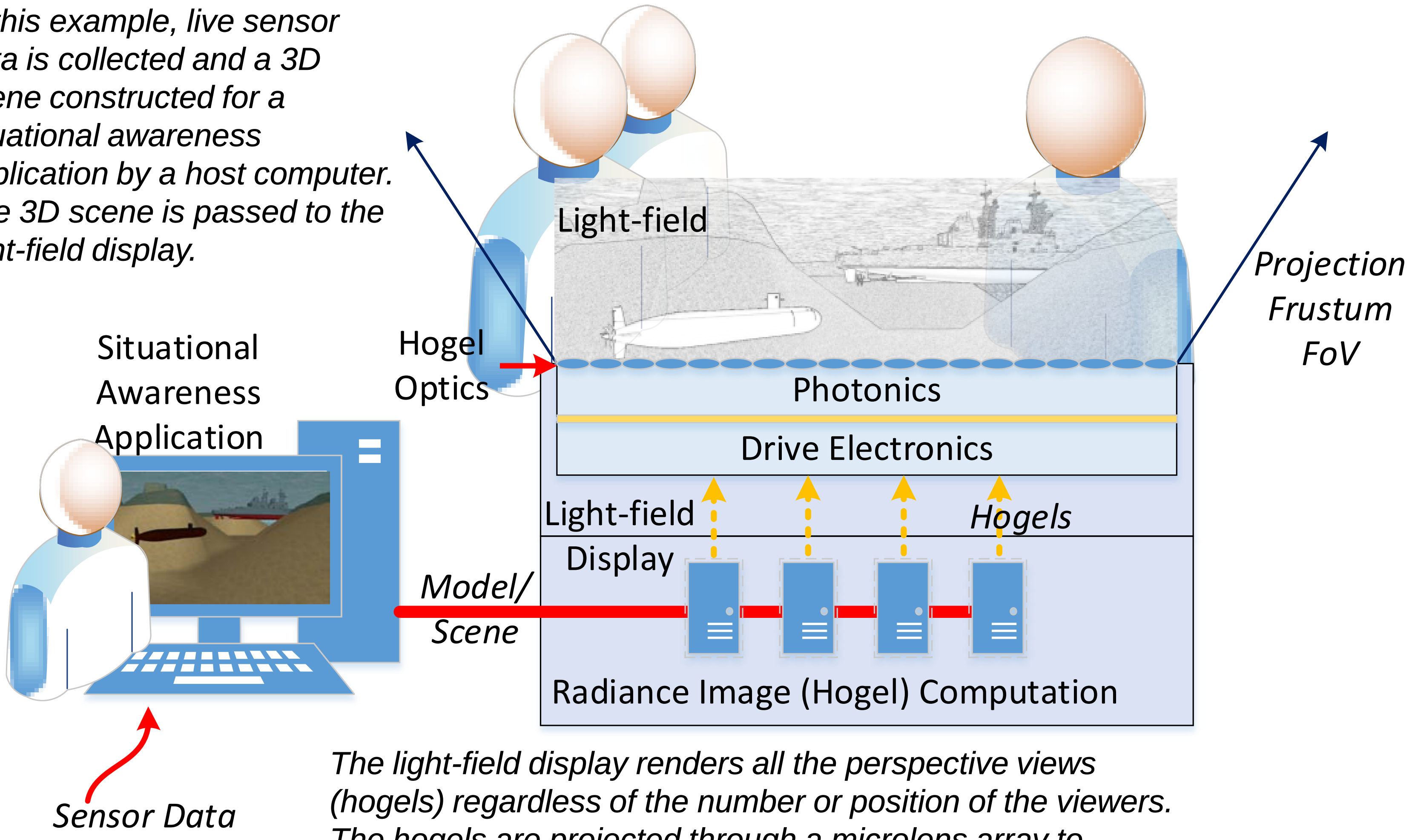
UPSD Prototype



http://www.fovi3d.com/light-field-display/
http://www.fovi3d.com/multiview-rendering/

Light-field Display Architecture

In this example, live sensor data is collected and a 3D scene constructed for a situational awareness application by a host computer. The 3D scene is passed to the light-field display.



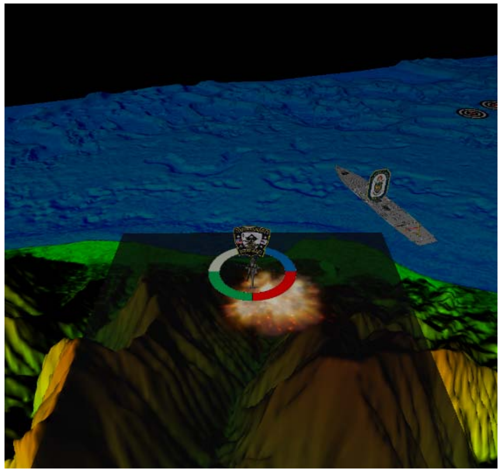
The light-field display renders all the perspective views (hogels) regardless of the number or position of the viewers. The hogels are projected through a microlens array to construct a 3D light-field for all viewers simultaneously. In essence, the light-field display is a large plenoptic projector.

FOVI 3D

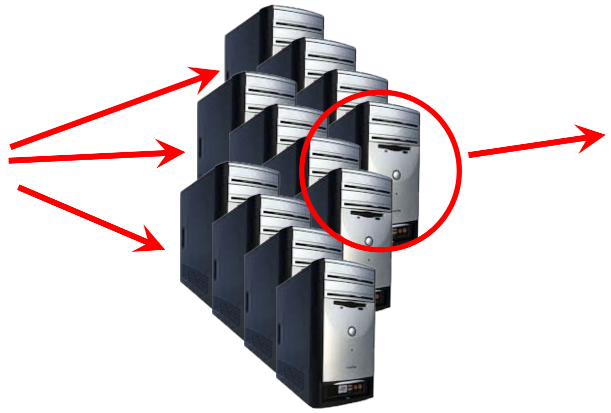
Light-field Radiance Image Rendering

Light-field radiance image rendering is the process of rendering all the projected views for a light-field display for a given scene/model.

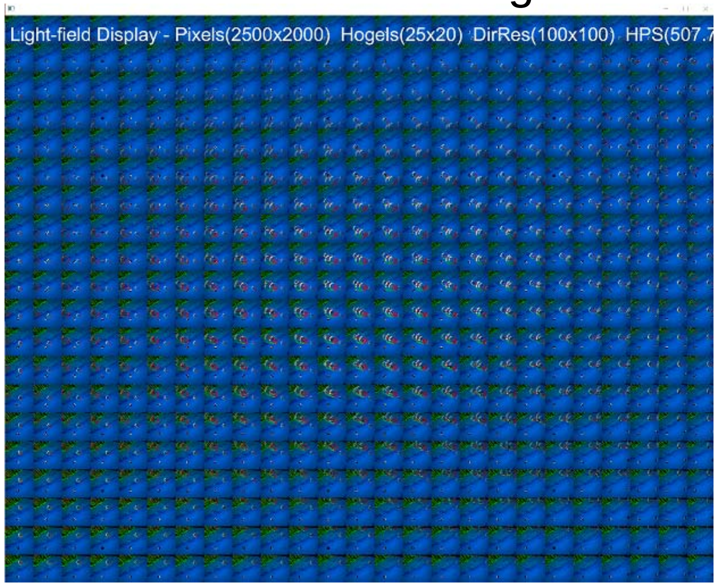
Host Application Scene



The scene is distributed to an array of PC/GPUs, each of which renders a subset of the light-field display radiance image



A subset of the light-field radiance image

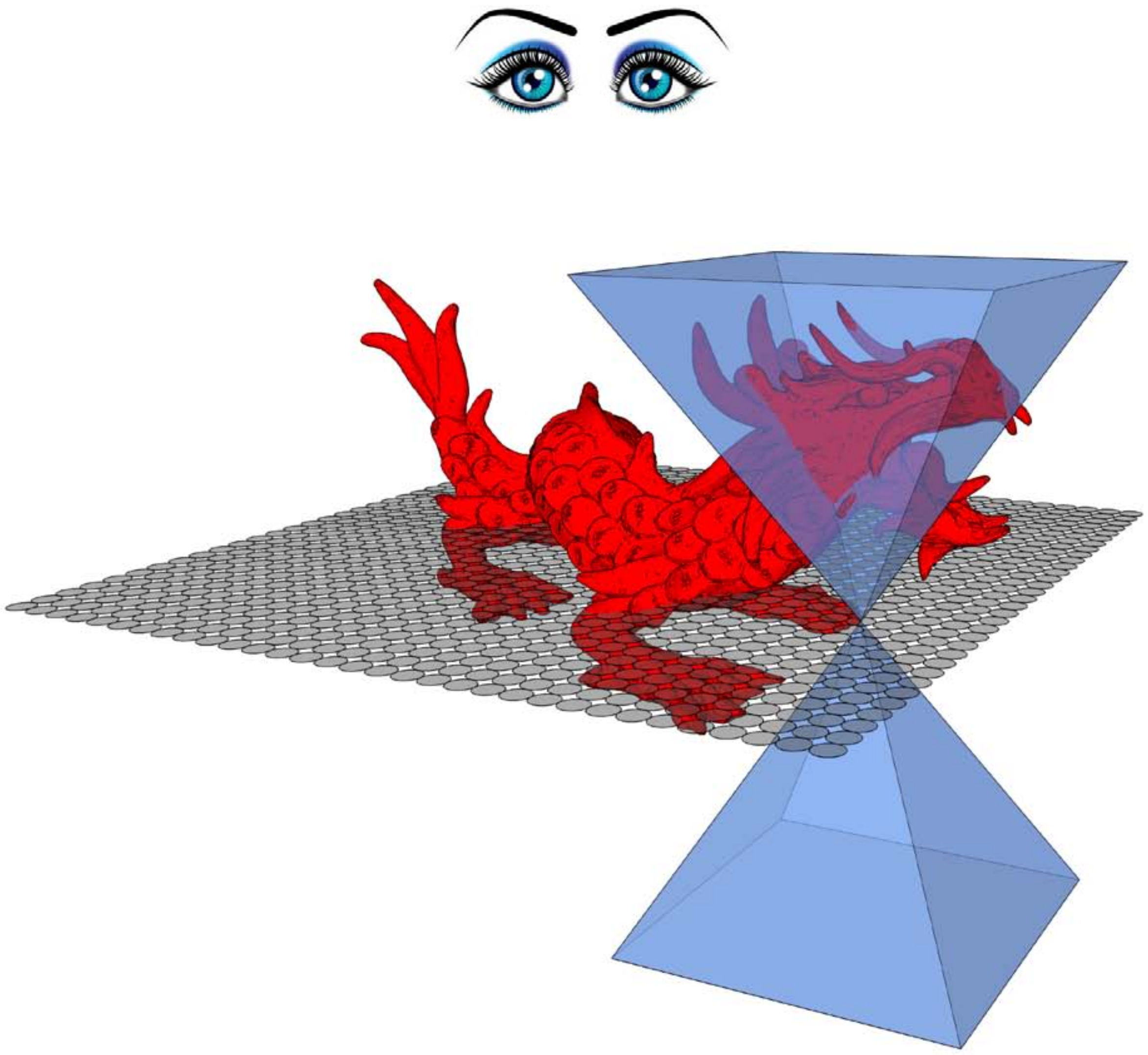

Light-field Display - Pixels(2500x2000) Hogels(25x20) DirRes(100x100) HPS(507.7

Requires:

- 3D model/scene
- Model of the display image plane; a mathematical model of the hogel lens array.
    - Hogel dimensions
    - Number of hogels
    - Position/orientation of hogels
    - Hogel FoV

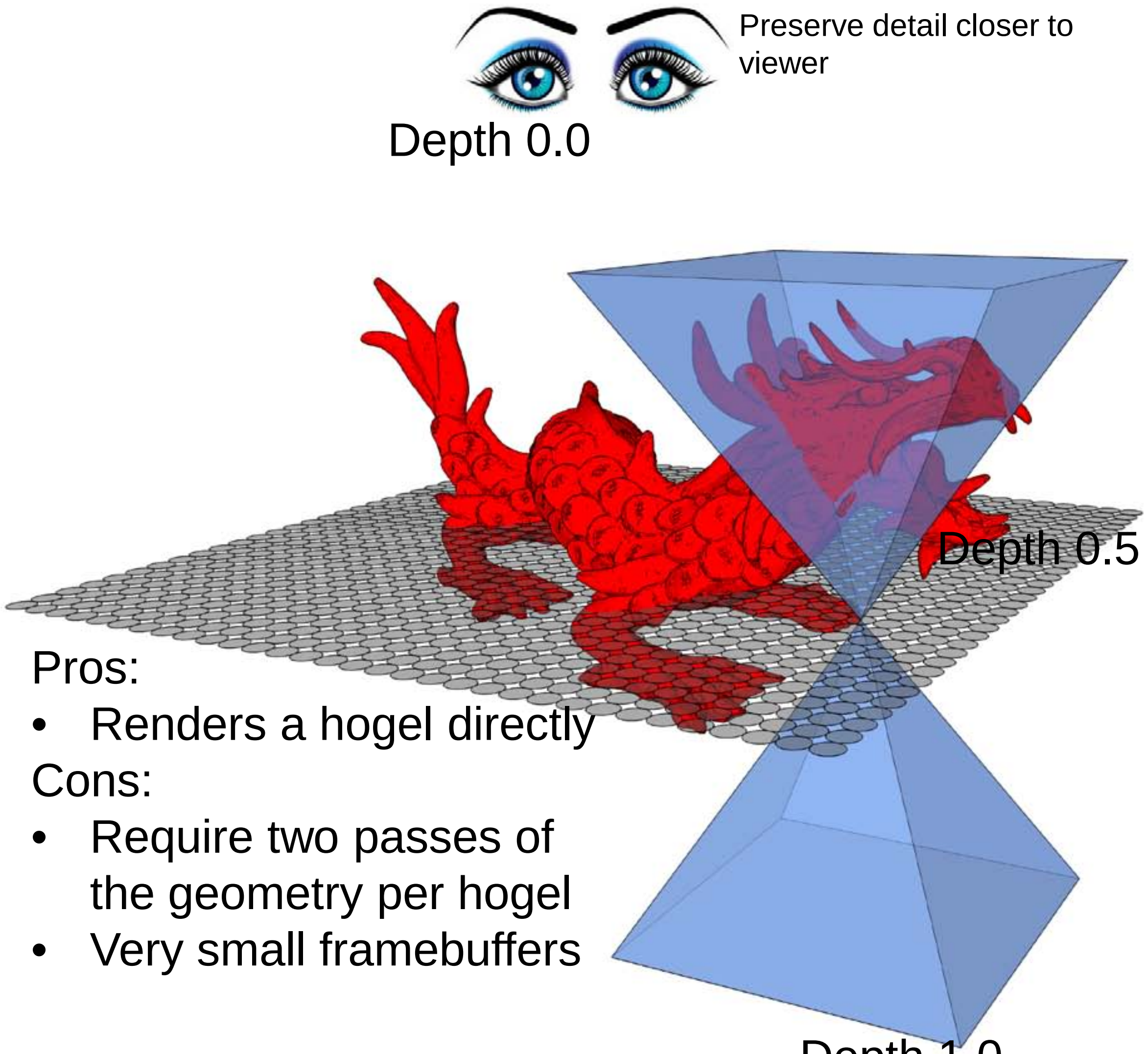
Light-field Display interactivity and update rate are proportional to the complexity of the scene/model, the power/configuration of the rendering cluster and the size of the light-field display radiance image. The Size, Weight, Power and Cost (SWaP-C) of radiance image computation limits integration of the light-field display.

Radiance Image Rendering – Double Frustum Rendering



- The hogel image plane is a mathematical model/description of the light-field display image plane defined in model/scene space. In essence, the hogel is a micro-image in a plenoptic/light-field radiance image. A hogel however, can represent 'light' on both sides of the image plane.
- The hogel image plane can be placed anywhere within the scene and as such can bisect a model.
- The projected hogel frustum defines the light that would emanate from the center of each hogel.
- This 'light' can represent model/scenery below the image plane.
- When raytracing, the ray starts from the outside of the scene and passes through the center of each hogel, creating a 'bowtie' hogel frustum.
- When double frustum rasterizing, the camera is placed on the hogel center and rendered downward, then the camera is flipped and rendered upwards without clearing the depth buffer. The upward camera is rendered preserving triangles farthest from the camera, thus closer to the viewer. When the two views are combined via their depth map the hogel 'bowtie' frustum is created.

Radiance Image Rendering – Double Frustum Rendering

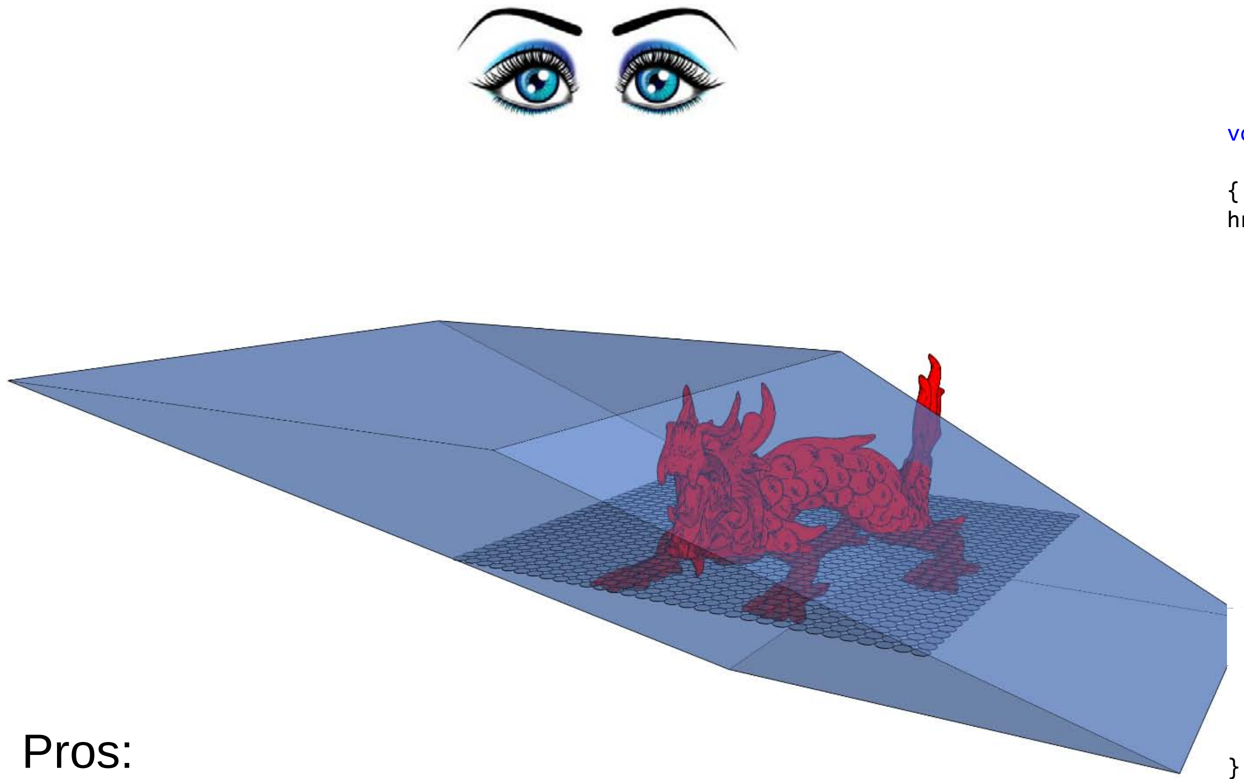


Pros:
- Renders a hogel directly

Cons:
- Require two passes of the geometry per hogel
- Very small framebuffers

```
void RenderBase::render(RenderState &rS,
                        const Hogel &hgl,
                        const CmdManager *pCmdManager)
{
glm::vec3   vE    = hgl.vE();
glm::vec3   vD    = glm::normalize(hgl.vU());   // render up the hogel normal
glm::vec3   vU    = glm::normalize(hgl.vD());
glm::ivec2  vC    = hgl.vC();
float       zF    = rS.zFar();
float       zN    = rS.zNear();

 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

 glDepthFunc(GL_LESS);
 glFrontFace(GL_CCW);

 // back frustum, away from the viewer into the display
 if (1)
 {
 CmdLst::const_iterator jj      = pCmdManager->cmdLst().begin();
 CmdLst::const_iterator jEnd  = pCmdManager->cmdLst().end();

  rS.backFrustum(vE + glm::vec3(0,0,zN),-vD,-vU);

  glCullFace(GL_BACK);
  glDepthRange(0.5f,1.0f);

  while (jj != jEnd)
    (*jj++)->exec(rS);

  glFlush();
 }

 // front frustum, toward the viewer out of the display
 if (1)
 {
 CmdLst::const_iterator jj      = pCmdManager->cmdLst().begin();
 CmdLst::const_iterator jEnd  = pCmdManager->cmdLst().end();

  rS.fowardFrustum(vE + glm::vec3(0,0,-zN),vD,vU);

  glCullFace(GL_FRONT);
  glDepthRange(0.5f,0.0f);

  while (jj != jEnd)
    (*jj++)->exec(rS);

  glFlush();
 }
}
```

Radiance Image Rendering – Oblique Slice & Dice Rendering



```
void renderObliqueView(hr::Camera &camera,
                       const glm::vec2 &rA)
{
hr::Control control;

  glClear(GL_COLOR_BUFFER_BIT |
          GL_DEPTH_BUFFER_BIT);

  glEnable(GL_DEPTH_TEST);
  glEnable(GL_CULL_FACE);

  {
  glm::mat4 mS;

    mS[2][0] = -glm::radians(rA.x);
    mS[2][1] = -glm::radians(rA.y);

    camera.warpView(mS);
    renderManager.render(camera,control);
  }

  glFinish();
}
```

Output Goes to Slicer.

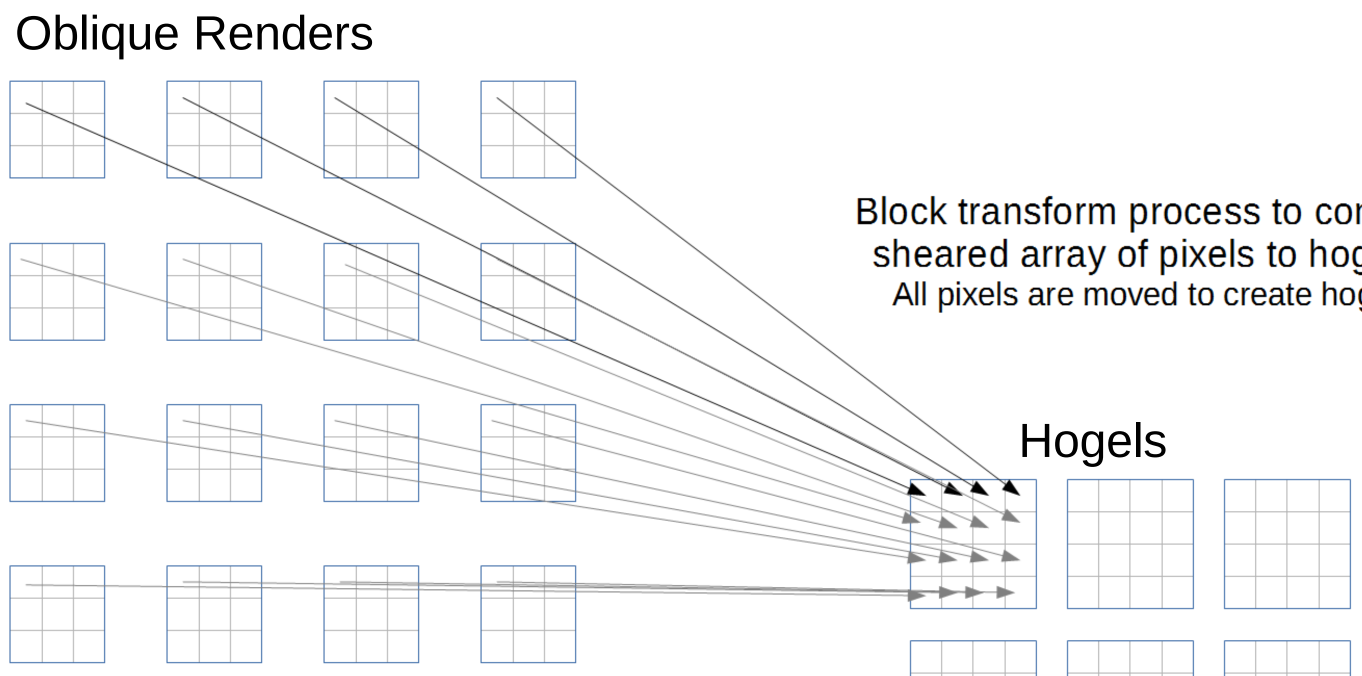
Pros:

- Large framebuffers, more efficient use of OTS GPUs

Cons:

- Requires conversion from oblique pixel space to hogels.
- Have to store, manage or re-render all the oblique images for slicing

Radiance Image Rendering – Oblique Slice & Dice Rendering



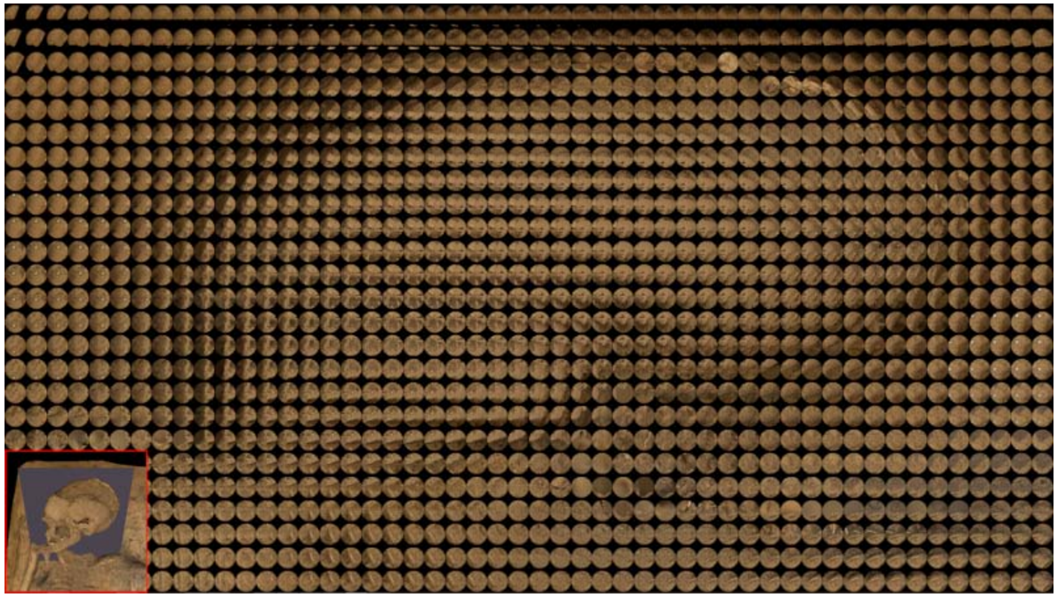
Block transform process to convert ortho sheared array of pixels to hogel views.
All pixels are moved to create hogel views.

Ortho sheared array of ray traced images
3x3 x 16 pixels = 144 pixels

Hogel views - the Result of block transform
4x4 x 9 =144 pixels

SWaP-C: The Cost of Double Frustum Light-field Rendering

This render example shows a 50 x 25 array of (76 x 76 pixel) hogels rendered from a burial scan model from the Smithsonian collection onto a 4K monitor with a NVidia GTX Titan. Triangles are batched, with bounding volumes and with camera frustum culling enabled; double frustum rendering at ~100 hps (~200 fps).



2K video: https://youtu.be/QCRXLzAThYQ

50x20 Array	76x76 Dir. Res.
Hogels	1,250
SpDU	12.5
500x500	**76x76 Dir. Res.**
Hogels	250,000
SpDU	2,500
Size	1.444 Gpixels
500x500	**512x512 Dir. Res.**
Hogels	250,000
Size	65.635 Gpixels

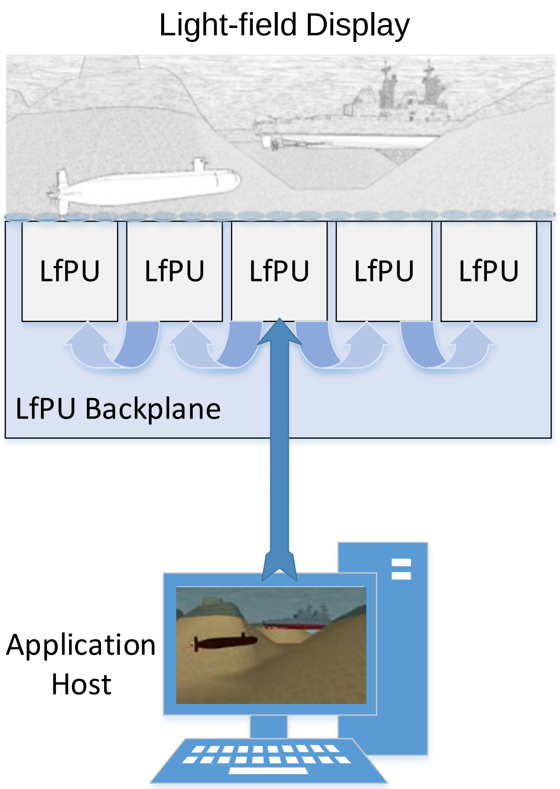
Vertex transform bound…. Why??? We are making thousands of render calls serially to very small viewports… serially....

Problem with using OpenGL and Modern GPUs for Light-field Rendering

- Many developers of novel display architectures have (or had) developed fix function OpenGL shims to intercept draw commands and forward them to a display specific renderer.
  - However, there is no agreement on which OpenGL version and/or functions to shim.
  - Shaders make shimming with confidence nearly impossible or is so highly restricted to the point where shaders may just emulate a fixed function pipeline with little variation.
  - There usually exists additional display specific extensions or APIs which then reduce the portability even more.
- Modern OpenGL has a single active viewport and a single active view matrix
  - Multi-viewpoint rendering becomes a responsibility of the host application which must cache the render commands and then re-render the list for each viewpoint.
  - Meanwhile, the host application has stalled until all the views are rendered because the render state can not change until all the views are rendered.
- The application developer has to know a lot about the target display.
  - Number of views
  - View specific projections and transforms
  - Distortion shaders

Light-field (Multi-View) Processing Unit (LfPU/MvPU)

The LfPU removes the dependency of using an array of OTS computers for rendering the radiance image, greatly reducing the associated SWaP-C constraints for integration.



The LfPU is

- Highly parallel radiance image rendering device
- Separated from the host CPU (or GPU) by an expandable, interconnect framework and provides many views (hogels) into a scene per scene frame
- Physically located in close proximity to the modulation layer and has direct write access to the modulation driver back-buffers
- The host application has no concept of light-field rendering. The host application provides a view volume from which the image plane is derived.

This reduces the complexity of a LfPU interconnect framework and removes the pixel transport bandwidth requirements required from OTS computation.

The LfPU interconnect framework will provide scene, command and sync buffering and relay throughout the topology.

Neither the host system nor the individual LfPUs would have knowledge of the interconnect topology or even the depth and breadth of the system.

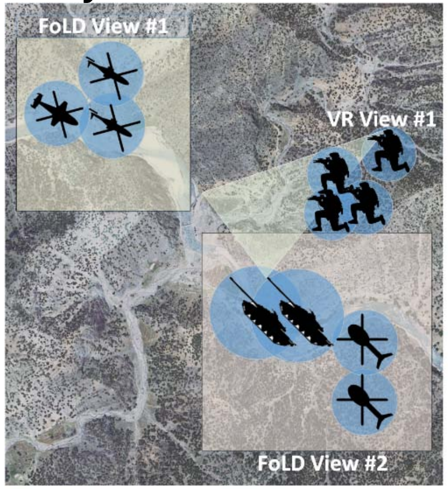
Results in a display architecture ready for application integration without a cluster of OTS render boxes.

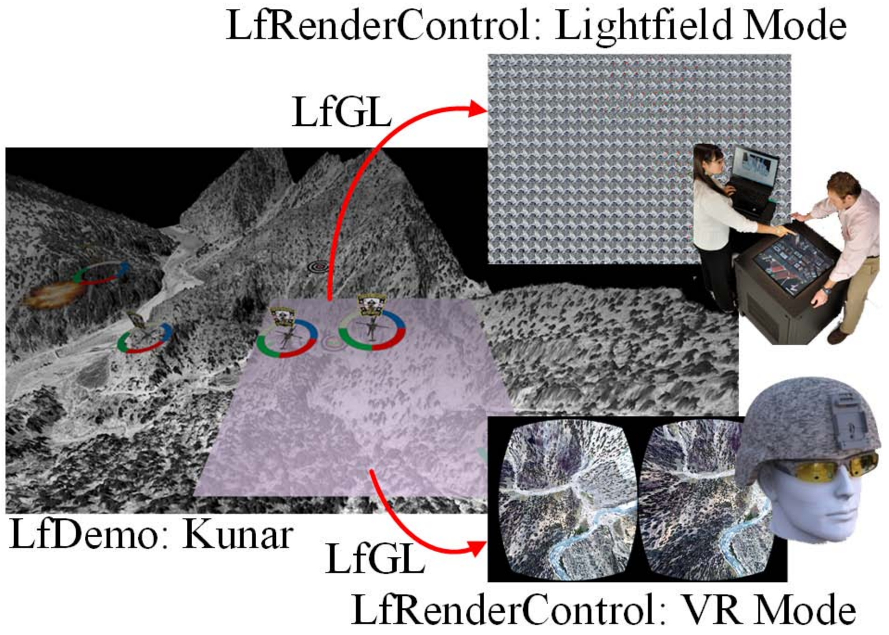
MvPU Graphics API for Light-field Rendering – Where to Start?

- First, just to get started, reduce the complexity of the rendering API while providing enough support for general purpose rendering:
    - Battlespace
    - Medical
    - Oil & Gas
    - CAD
- Therefore, what is the minimum required graphics definition to provide natural depth cues: Occlusion, gradient shading, specular highlights, binocular disparity.
- Minimal light-field rendering
    - Phong shading
    - Texture mapping
        - Strict material definition
    - One vertex list definition
        - Vertex, Normal, TexCoord
    - Support for transparency (1 or 2 blend modes?)
- Add high level support for accelerated/advanced display rendering
    - Segmentation into background and foreground geometry
    - Bounding volumes for view culling as part of the vertex list definition
    - Build in support for dynamic geometric/texture level.
    - Provide cache management and synchronization for independent display render frame rates.
- Require pre-caching of texture and geometry before render cycles.   Changing the geometric/texture definition is not permitted during rendering.

Heterogeneous Visualization

Physics Simulation





- The host application defines a render view volume, not a single view point.
- The render code behind the API queries the display device for the correct render transforms, viewpoints, etc.
- Rendering won’t be in a ‘cloud’.
    - Implies that the cloud knows too much about the downstream device.
    - Very costly to send images.
- Physics may be in a ‘cloud’, geometry will be cached local to the device.
    - Implies that draw commands will be handles and transforms – bind and render
    - Devices will provide views within a global application/simulation world
    - Any distortion corrections/shaders are a function of the display device.