Object Graphics Library (ObjGL) and the Heterogenenous Display Environment

October, 2017

Introduction

Thomas Burnett

CTO, 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

Strong technical team with deep experience in optical, mechanical, electrical and software engineering.

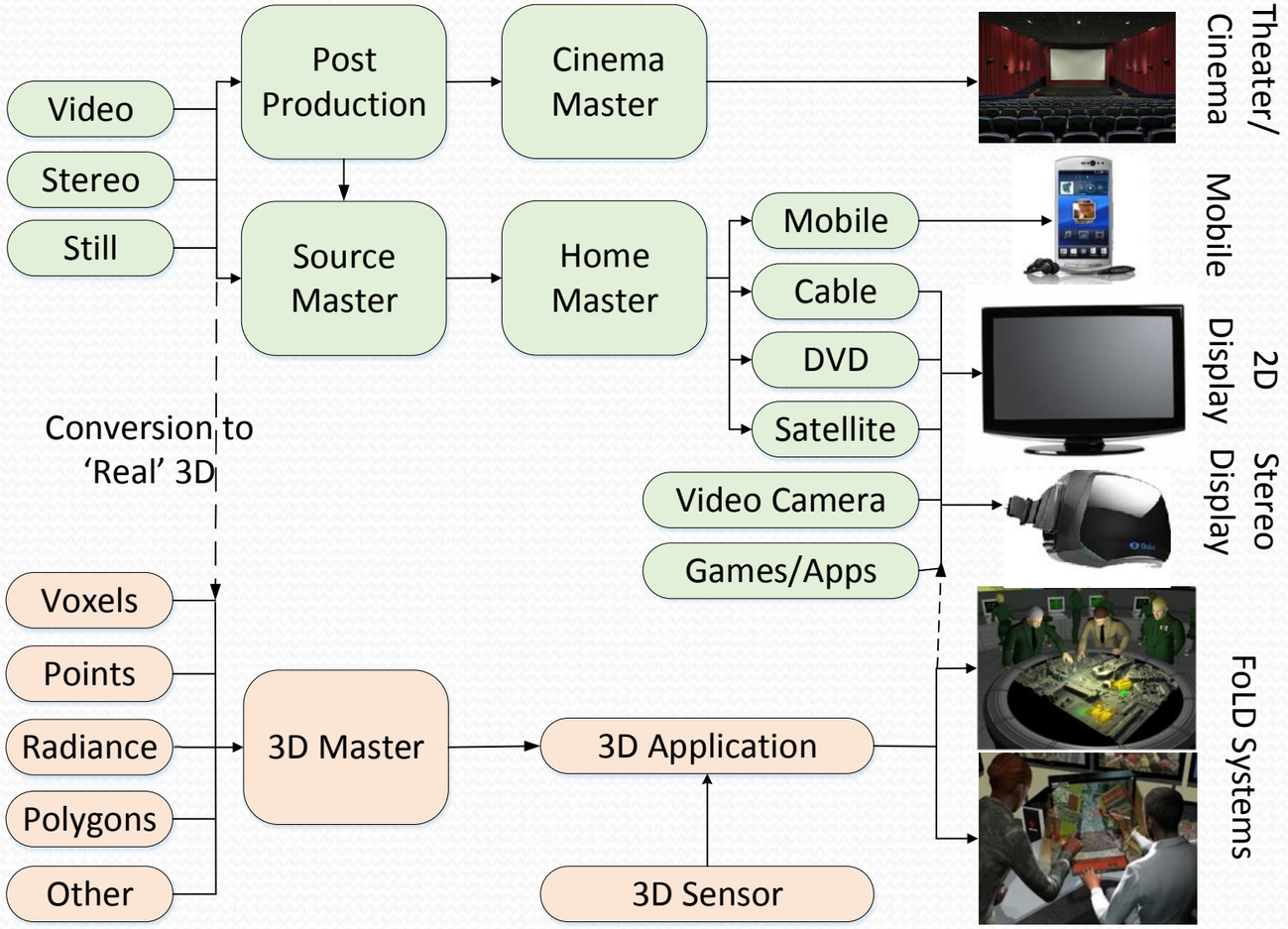
Agenda

- Review of the Challenges Posed by Streaming Media for Field of Light Displays
- Review of Two FoLD Systems
- Issues with OpenGL and GPUs for Multi-view Rendering
- Object Graphics Library and the Heterogeneous Display Environment

Review of the Challenges Posed by Streaming Media for Field of Light Displays

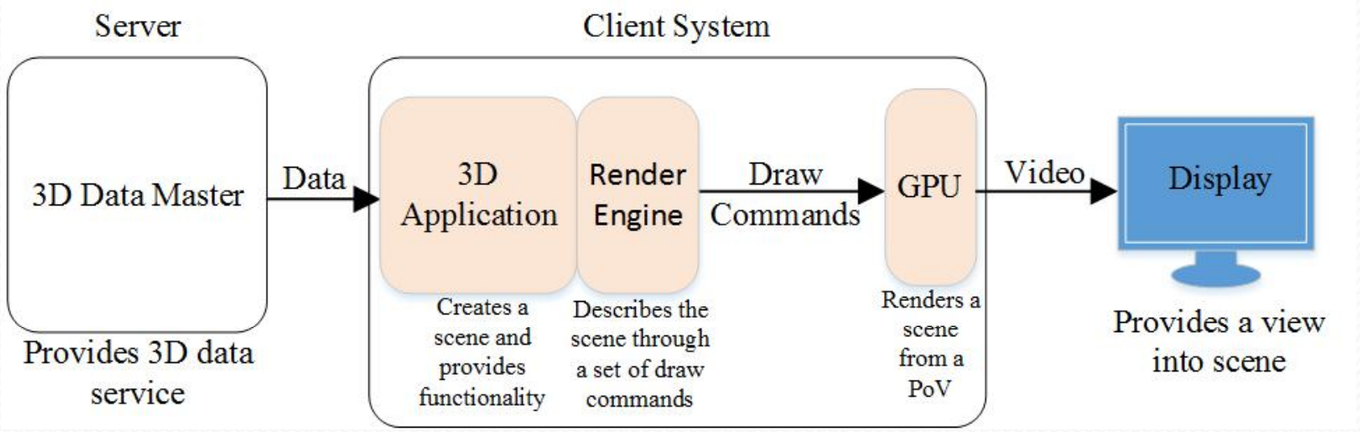
Source Master and the FoLD System

1. 2D video is captured with the expectation that the downstream display offers a single point of view.
2. 3D visualization requires actual 3D real-world coordinates in three-dimensional space.

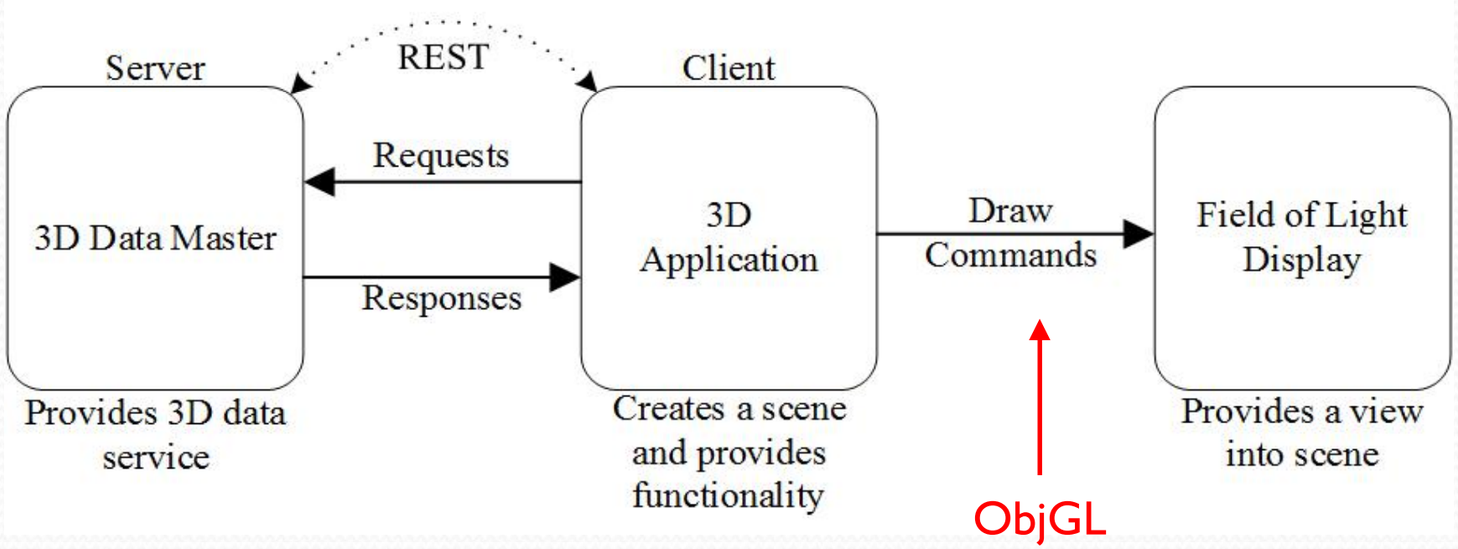


Typical Application Data Rendering Flow

1. The client system, thus the 3D application/render engine is tightly bound to display particulars.
2. The 3D application/render engine controls the GPU which renders a video stream to the display.
3. If the display architecture changes, the 3D application/render engine and possibly the interface require modification



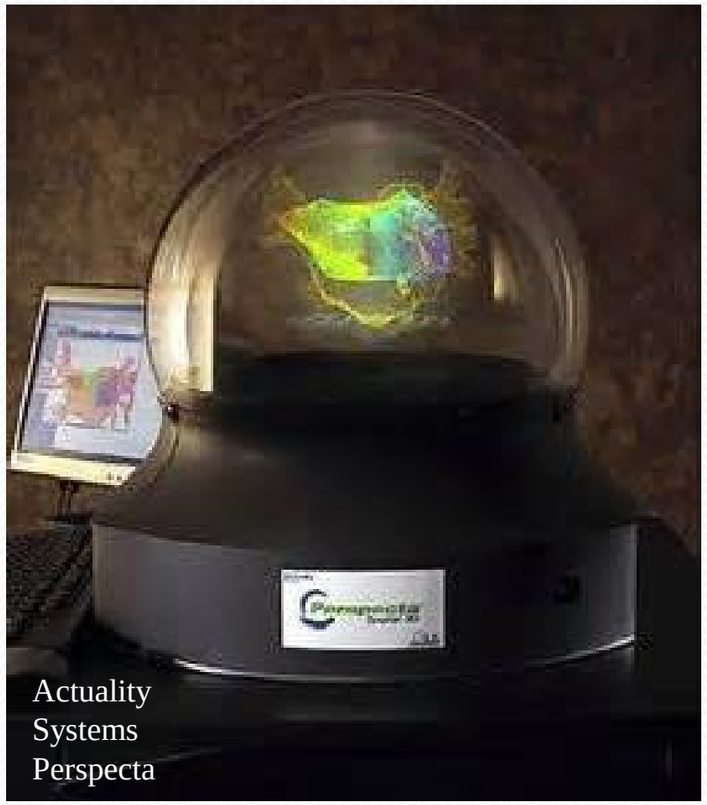
The Two SMFoLD Challenges



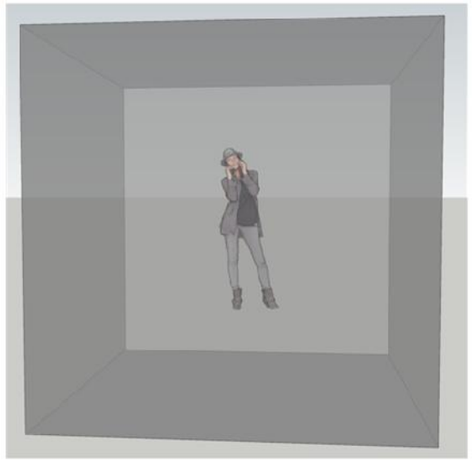
1. The client/server model and protocol that describes the relationship between a service provider (server) and a consumer (client).
2. The display agnostic draw commands. ObjGL is intended to solve this challenge.
   a) Architecture unknown
   b) Location may be remote

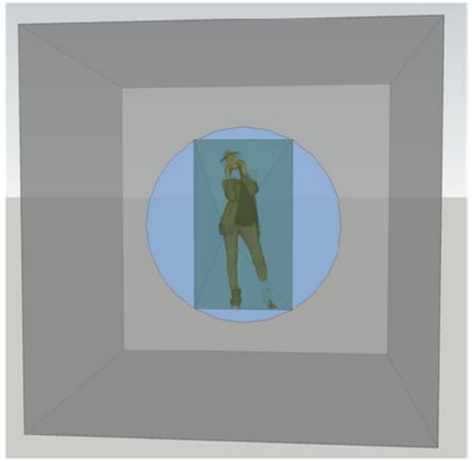
Review of Two FoLD Systems

Volumetric Rendering



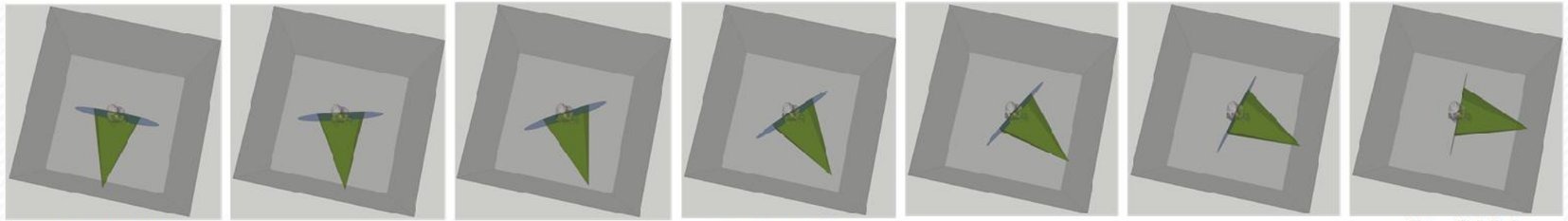
Want to draw a scene in the middle of a unity cube.





Need a diffuser and a projection frustum

The renderer needs to know the camera projection matrix and the position and orientation of the camera (in the form of a 4x4 transform) in normalized 3D display space.



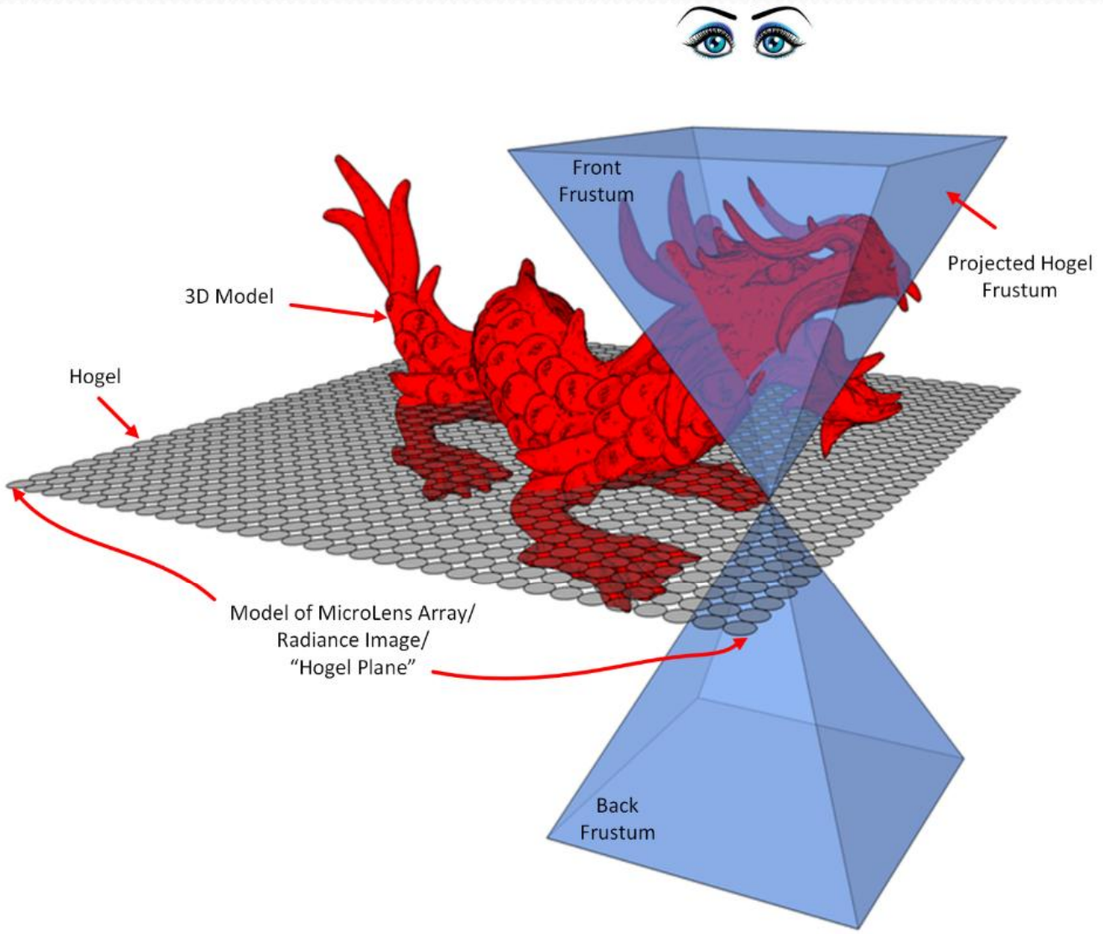
cPos: 0,0,0.5
cDir: 0,0,-1
cUp:0,1,0

cPos: 0.35,0,0.35
cDir: -0.7,0,-0.7
cUp:0,1,0

cPos: 0.5,0,0
cDir: -1,0,0
cUp:0,1,0

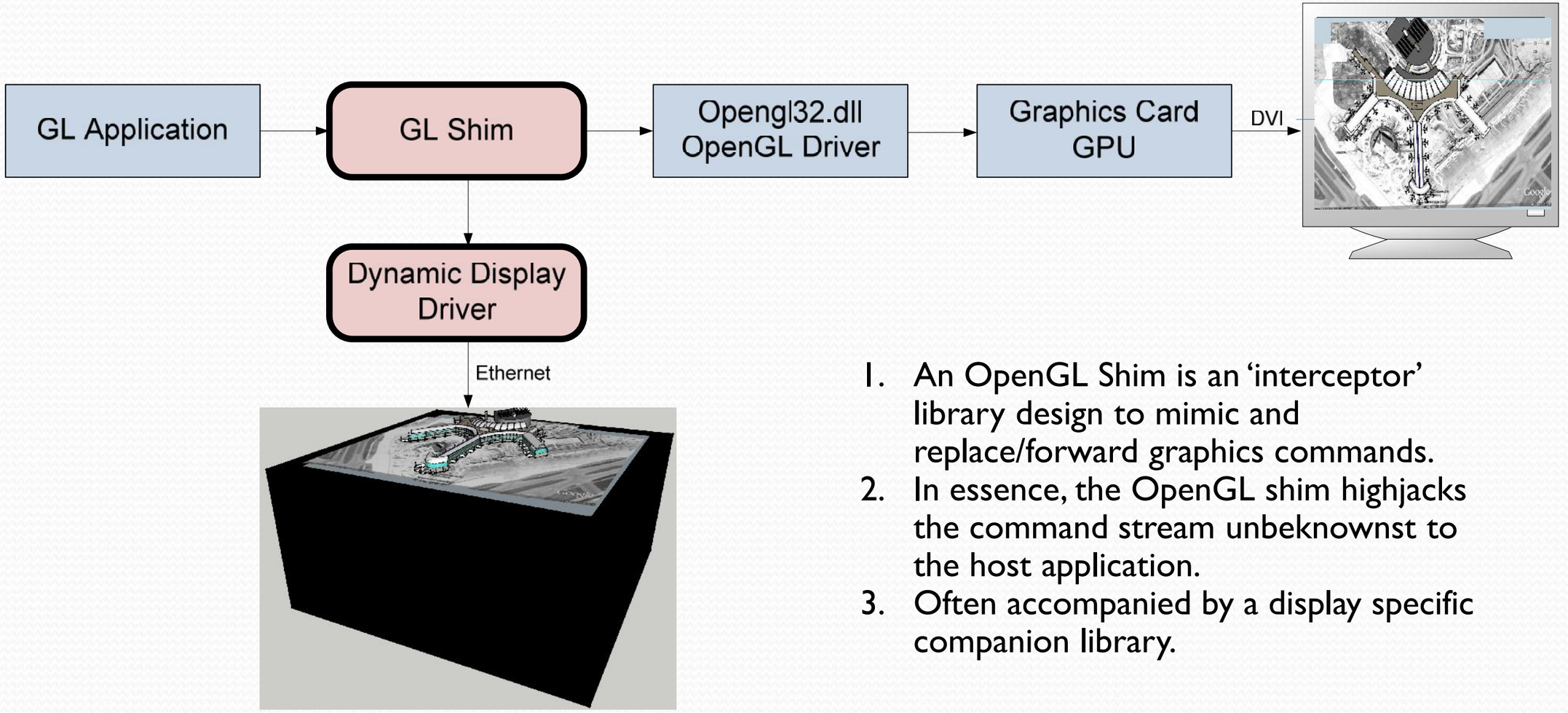
Light-field Rendering

Zebra Imaging
ZScape Motion Display



Issues with OpenGL and GPUs for Multi-view Rendering

The OpenGL Shim

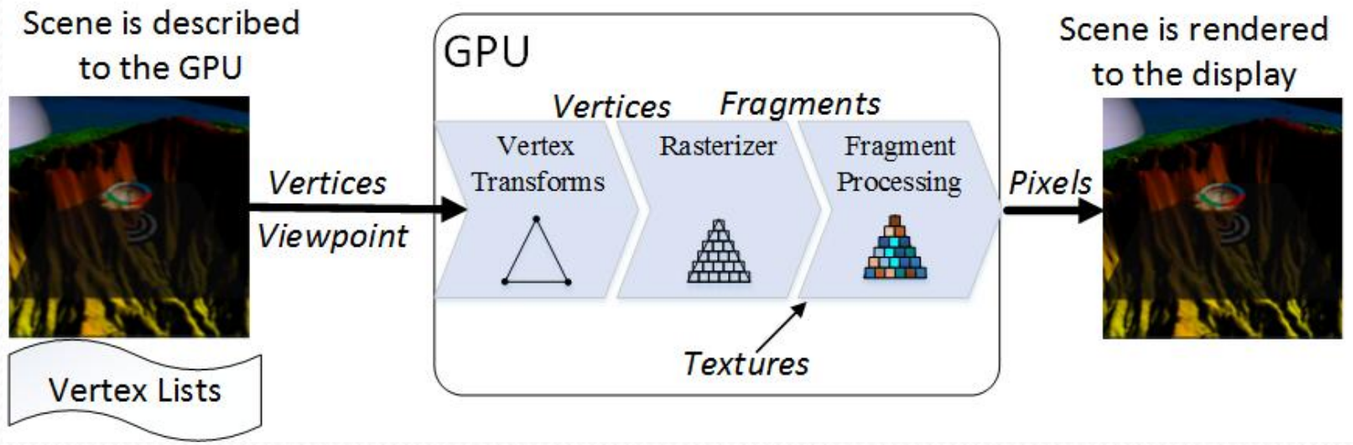


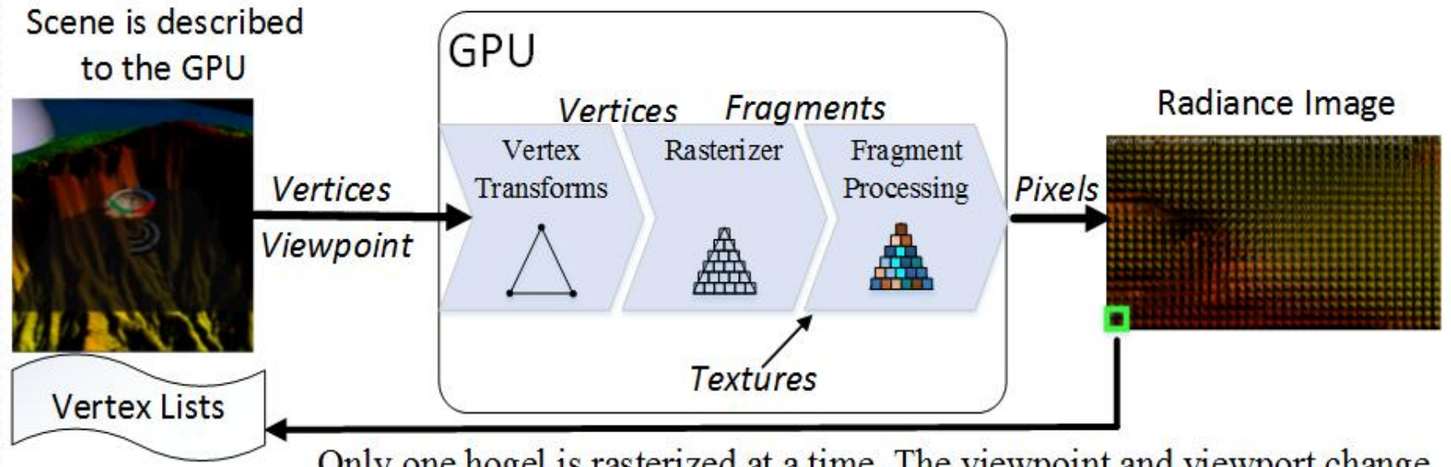
1. An OpenGL Shim is an 'interceptor' library design to mimic and replace/forward graphics commands.
2. In essence, the OpenGL shim highjacks the command stream unbeknownst to the host application.
3. Often accompanied by a display specific companion library.

Issues with the OpenGL Shim Concept

- There is no agreement among novel display developers on which OpenGL version and/or functions to shim. Generally, most interceptor libraries are focused on the legacy fixed function pipeline APIs defined in the OpenGL 1.0-1.5 specifications. Some of the more optimal interceptors can support later versions of OpenGL; however, the programmable nature of later OpenGL specifications are more difficult to shim. Shaders make intercepting/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. In either event, intercepting OpenGL commands is typically regarded as intercepting the fixed function APIs which are considered legacy commands by most modern developers.
- There are legacy fixed function APIs which are suboptimal for specifying geometry that programmers often default to because of their ease of use and readability. These immediate mode functions perform poorly in the context of multi-view rendering. Their counterpart functions, the retained mode APIs, are far better suited for multi-view rendering; however, the retained model APIs have evolved significantly over the course of the OpenGL specification and not every application is implemented with the best constructs; thereby reducing render performance.
- Many of the OpenGL functions and functionality are not applicable to distributed or multi-view rendering. For example, point and line rendering is computed in screen space and is not 3D. This implies that any application that does native OpenGL point and line rendering cannot image points and lines for a light-field display; however, points do have the ability to be converted into billboarded polygons when properly specified.
- Fixed function OpenGL defines a single GL_MODELVIEW matrix which is the pre-multiplied model and view matrices. This implies that for any display that requires multi-view rendering, the view component needs to be removed from the GL_MODELVIEW matrix or that the view matrix is specified in some other manner.
- Modern OpenGL has a single active viewport and a single active view matrix; therefore, applications/render engines written in OpenGL can only render a single view to a single destination framebuffer at a time. There is no agreed upon way within the context of OpenGL to specify a multi-view render volume.
- Displays that require multi-view rendering often require much more time to complete a display frame than the host application. This requires that the host application stall and synchronize with the render cluster. Often this situation reduces the frame rate of the host application and sacrifices host application interactivity.
- 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. Therefore, the host application must understand the exact nature of every view rendered for the display as well as any distortion corrections for any display specific optical properties.

The Fixed Function Graphics Pipeline





Only one hogel is rasterized at a time. The viewpoint and viewport change per hogel but the scene description remains the same.
Vertex dispatch/transform dominates pipeline.

Magnitude of the Light-field Radiance Image Rendering

Display Size (mm) *1mm Hogels*	Directional Resolution (rays)	Display Frame (Pixels)	50% Compression	75% Compression	90% Compression
256 x 256	128 x 128	268,435,456	134,217,728	67,108,864	26,843,546
512 x 512	128 x 128	4,294,967,296	2,147,483,648	1,073,741,824	429,496,730
1,024 x 1,024	128 x 128	17,179,869,184	8,589,934,592	4,294,967,296	1,717,986,918
256 x 256	512 x 512	17,179,869,184	8,589,934,592	4,294,967,296	1,717,986,918
512 x 512	512 x 512	68,719,476,736	34,359,738,368	17,179,869,184	6,871,947,674
1,024 x 1,024	512 x 512	274,877,906,944	137,438,953,472	68,719,476,736	27,487,790,694

To calculate the number of pixels generated within a second in the case of an interactive display, the frame rate multiplied by the display frame size.

Challenge

- Huge number of pixels to transfer.
- Assumes some knowledge of the display capability.

Desire

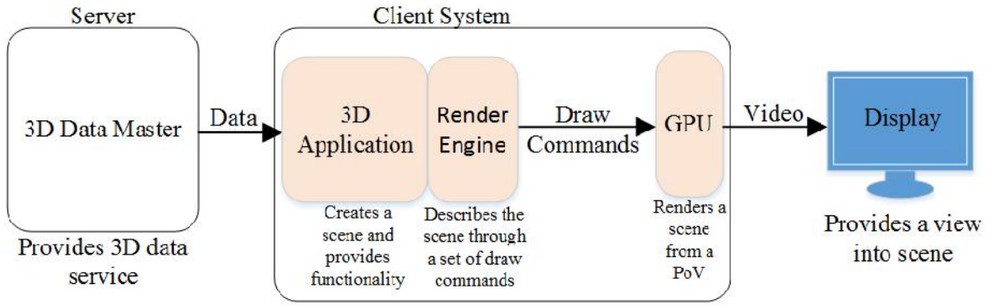
- Don't pass pixels!
  - Precache geometry
  - Send transforms
- Typical transform is a 4 x 4 matrix
- Typical object transform command would be in the order of 68 (16 * 4 + 4) bytes minimum

Another issue with the pixel data is that the light-field is generally captured/generated from a particular PoV which implies that there are possible occlusions within the light-field. If the light-field display projects from the point of view of the capture/generator, occlusions may not be apparent. However, as one moves about the scene, these occlusions become more apparent and appears as 'holes' where there is no pixel content to render. Since the nature of the FoLD rendering should be agnostic to the data, the best paradigm for display agnostic scene description is to transfer and render polygonal/model data

Object Graphics Library and the Heterogeneous Display Environment

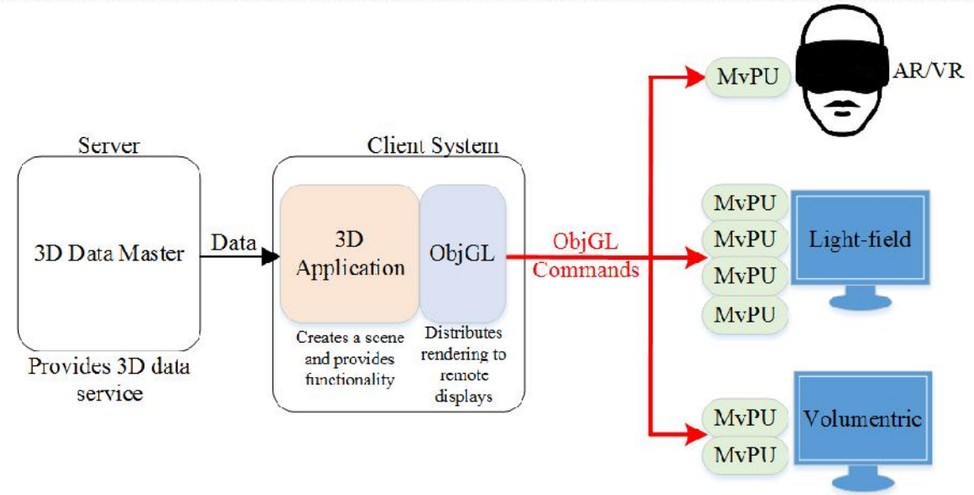
Object Graphics Library (ObjGL)

Current Display Rendering



Displays are data agnostic. The 3D application is responsible for fetching/reading data and describing the data by use of a 'Render Engine' to the GPU via a series of draw commands. The GPU in turn, renders a scene and produces a video feed to a display. The display is unaware of the video content.

Heterogeneous Display Environment



3D displays are data agnostic. The 3D application is responsible for fetching/reading data and describing the data using ObjGL to a heterogeneous display environment. The client application unaware of the rendering concerns of the 3D display. Each display has an array of Multi-view Processing Units (MvPU) that render display particular content. The 3D display is unaware of the nature of the content.

ObjGL is a FoLD (Field of Light Display) agnostic graphics application interface that can be used in a heterogeneous 3D display environment. It is intended to be a modern replacement for OpenGL (currently, the fixed function pipeline) yet with higher level functionality.

ObjGL Instructions: Control, Cache and Render

Control	Cache	Render
Clear	Cache Viewpoint	Activate Light
Finish	Cache View Volume	Bind Material
	Cache Light	Render VertexLst
	Cache Material	Deactivate Light
	Cache VertexLst	Bind Texture
	Cache Texture	Look
	Remove Viewpoint	Survey
	Remove View Volume	
	Remove Texture	
	Remove Light	
	Remove Material	
	Remove VertexLst	

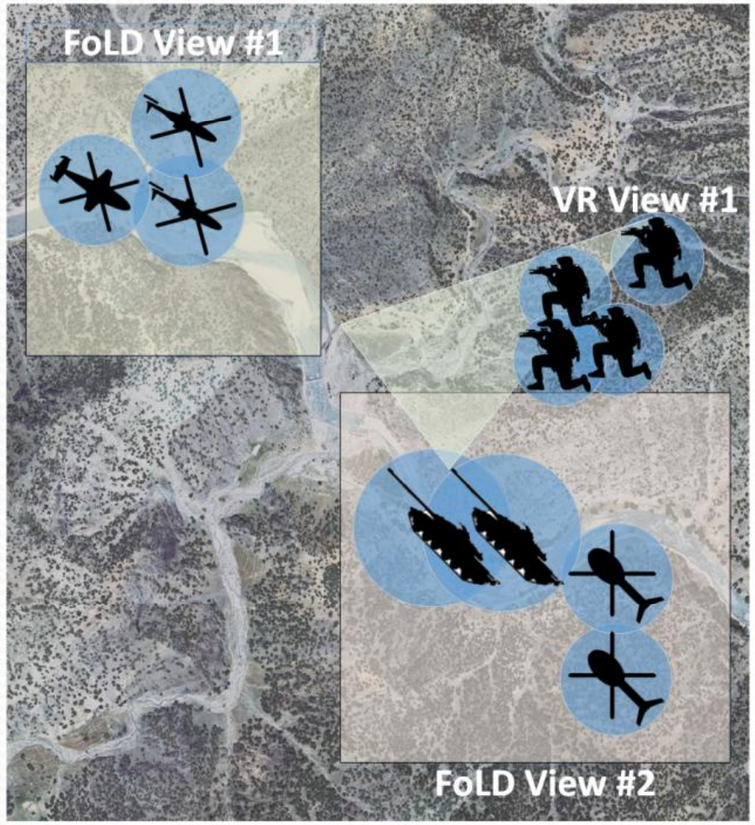
Cache Material
Cache Vertex List
Cache Light
Cache Viewpoint
Cache View Volume
- **Clear**
  - **Activate Light**
  - **Bind Material**
  - **Render Vertex List**
- **Finish**

Cache Material2
Cache Vertex List2
Update Viewpoint
- **Clear**
  - **Activate Light**
  - **Bind Material**
  - **Render Vertex List**
  - **Deactivate Light**
  - **Bind Material2**
  - **Render Vertex List2**
- **Finish**

Remove Material
Remove Vertex List
Remove Light
Remove Viewpoint
Remove View Volume

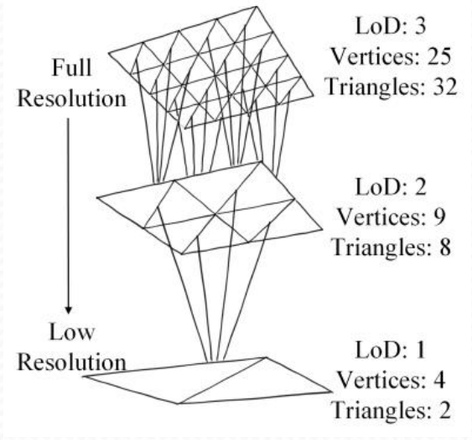
First-Person and Display Centric Views

- **First Person Perspective**
  The first-person perspective renders the scene relative to a single viewer. The first-person view is the normal view users expect when viewing any 2D video source or 2D/3D game on a standard 2D monitor.
- **Display Centric Perspective**
  The display centric perspective defines a position and orientation of a view volume whereby the scene is rendered from the perspective of the display (as opposed to the viewer). The view volume definition consists of the half-widths of a bounding rectangle and a model transform. A display that renders outward in a cylindrical or spherical sense from the center of a defined volume will require a display centric volumetric definition.

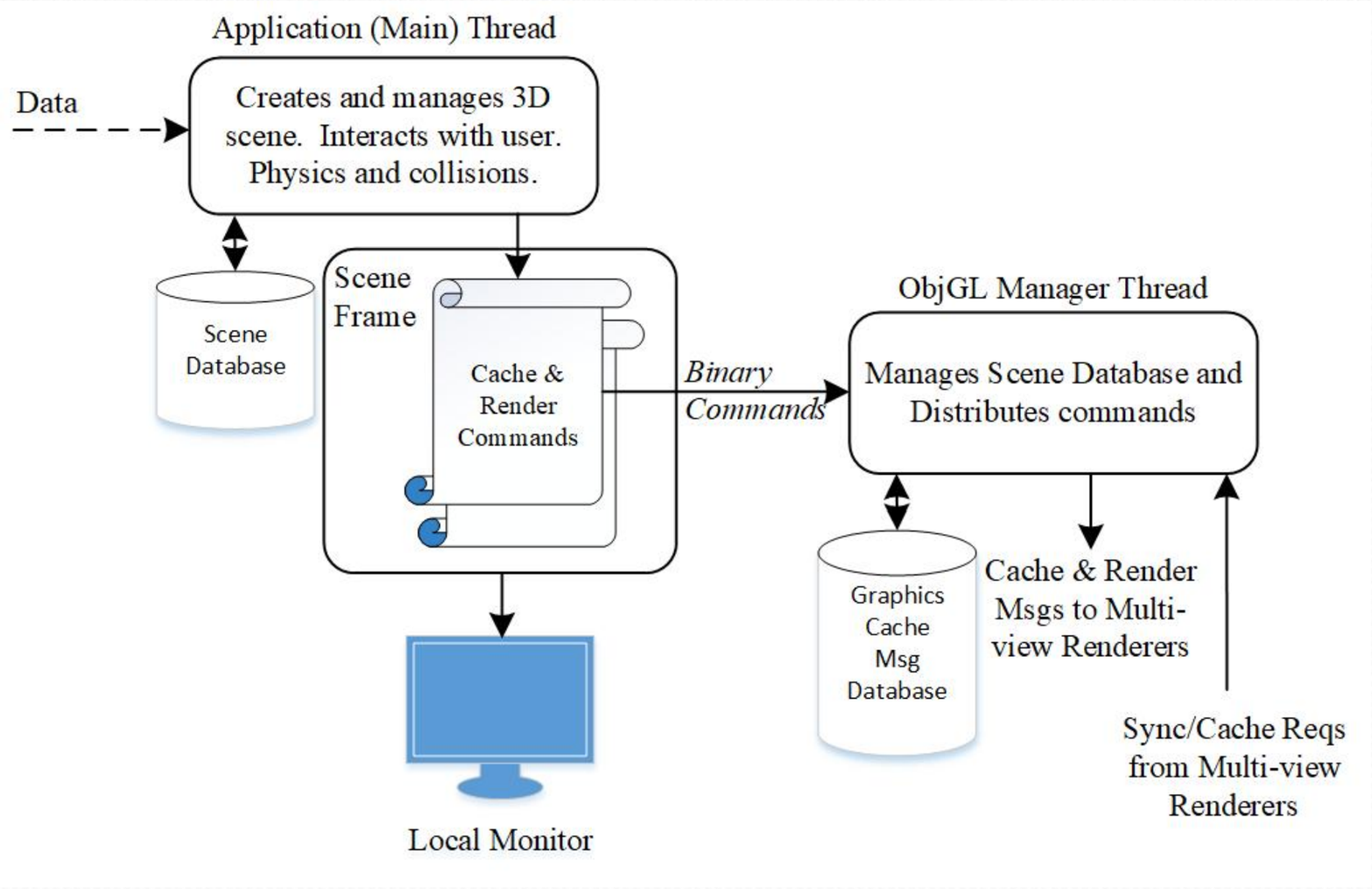


Render Acceleration Concepts

- Foreground/Background Segmentation
- Level of Detail
  - Data Phasing
- Criticality
- Bounding Volume

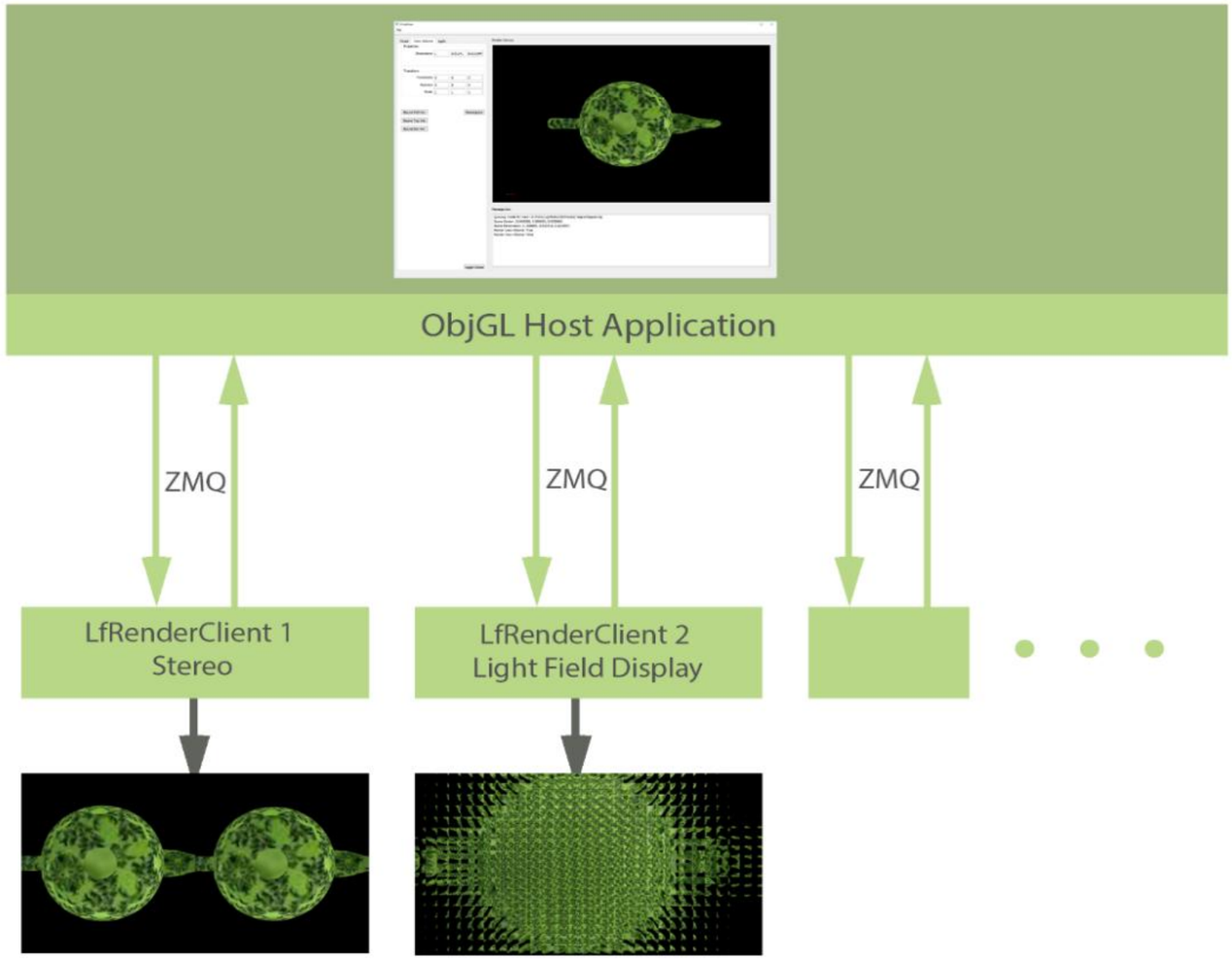


Application ObjGL Thread Model



Applications and Render Clients

Application



3D Displays

First-Person View Display Centric View

ObjGL Demonstration