Semester Thesis:

# Development of a Parallel OpenGL Polygon Renderer

Tobias Wolf

19. December 2007

# Outline

- **Introduction**
- Application Design
  - Equalizer
  - eqPly
- Data Structure
  - Old
  - New
- Rendering
- Summary

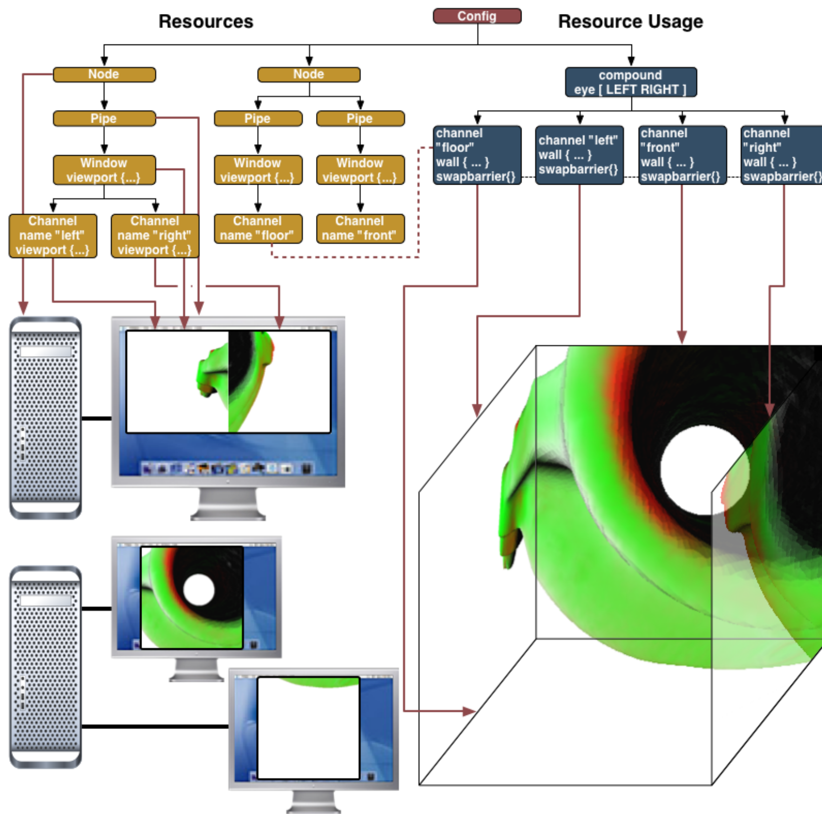
# Introduction

- Subject of this semester thesis
  - Development of an application for the efficient presentation of polygonal data based upon the Equalizer framework and OpenGL
- Sub goals
  - Development of an efficient data structure for the polygonal data
  - Implementation of an efficient and visually appealing rendering using Vertex Buffer Objects and GLSL shaders
- Old eqPly

# Outline

- Introduction
- **Application Design**
  - **Equalizer**
  - eqPly
- Data Structure
  - Old
  - New
- Rendering
- Summary

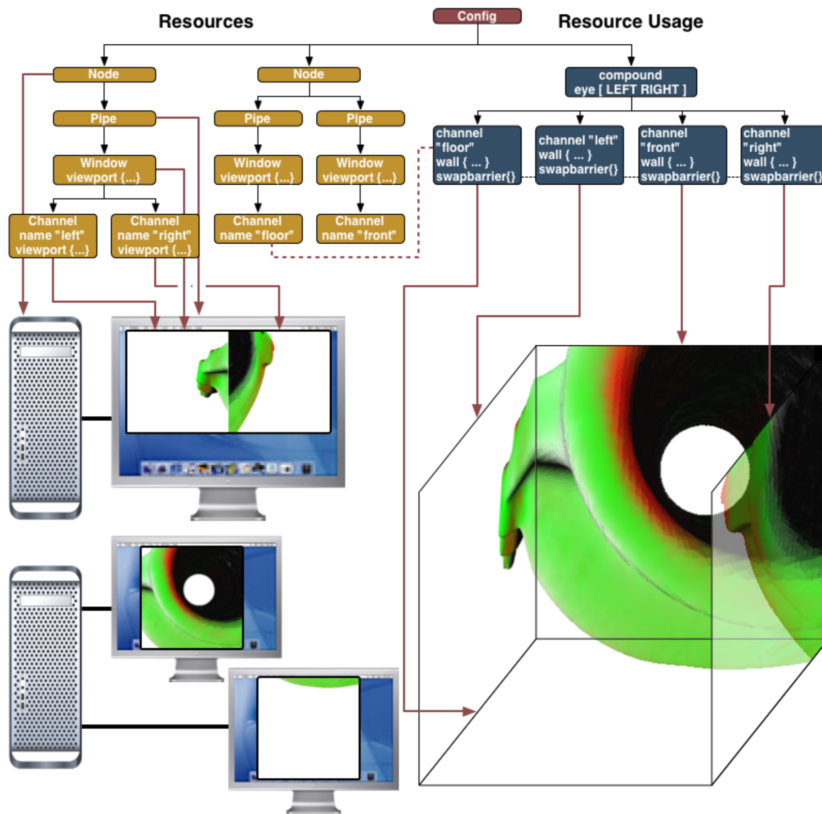
# Equalizer concepts and classes (1)




Equalizer sample configuration (source: Equalizer web site)

- Configuration
  - Description of physical and logical rendering resources
  - Description of resource usage

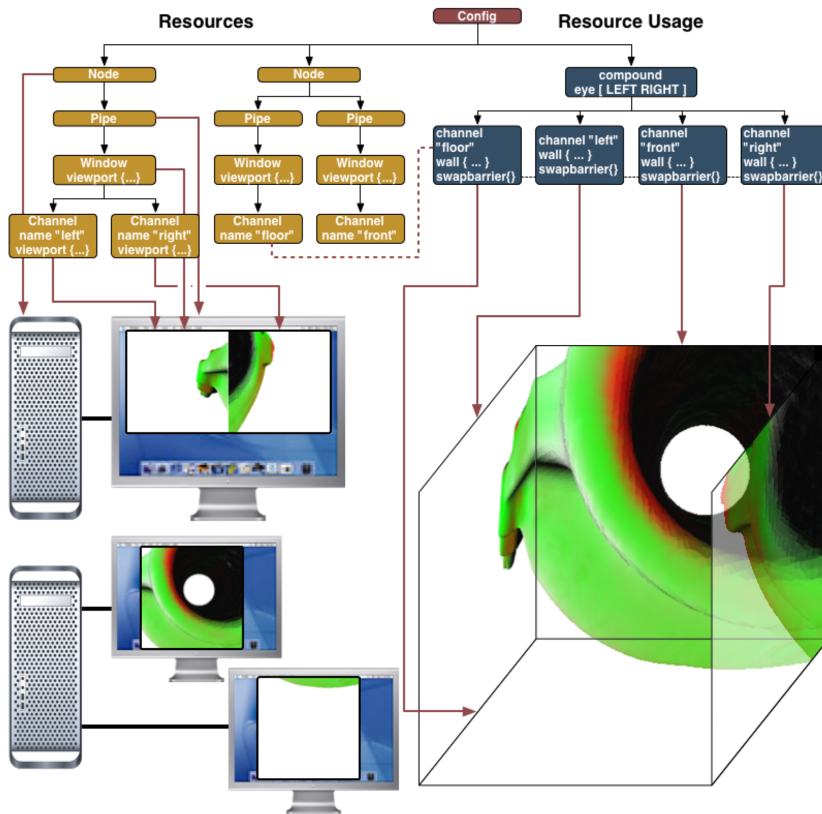
# Equalizer concepts and classes (1)




Equalizer sample configuration (source: Equalizer web site)

- Configuration
  - Description of physical and logical rendering resources
  - Description of resource usage
- Node
  - Representation of an individual computer in the rendering cluster

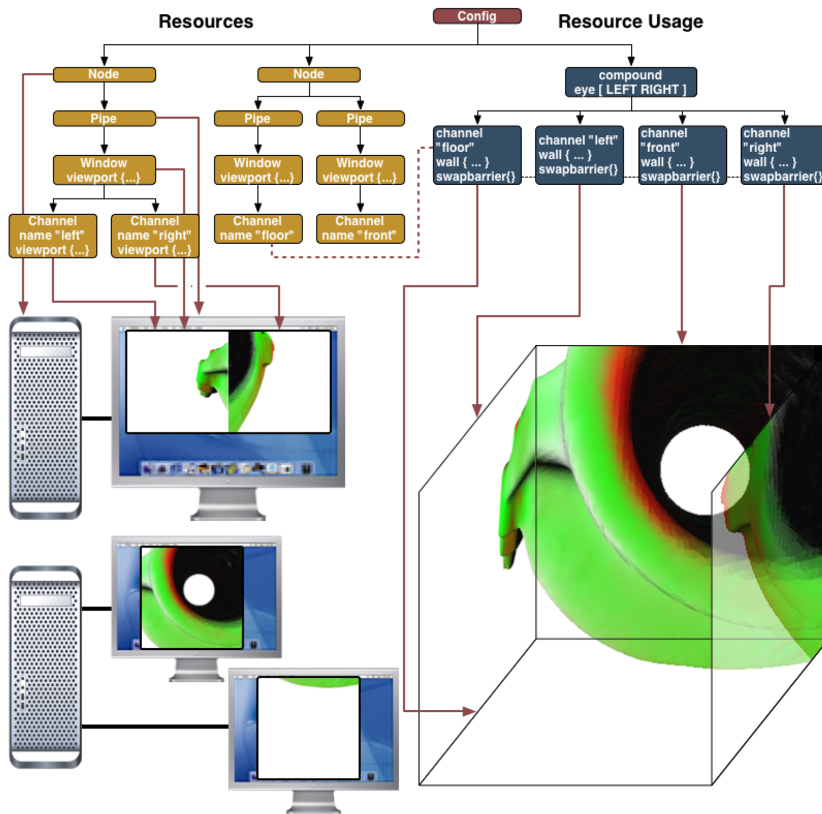
# Equalizer concepts and classes (1)




Equalizer sample configuration (source: Equalizer web site)

- Configuration
  - Description of physical and logical rendering resources
  - Description of resource usage
- Node
  - Representation of an individual computer in the rendering cluster
- Pipe
  - Representation of the graphics cards (GPUs) of a single node

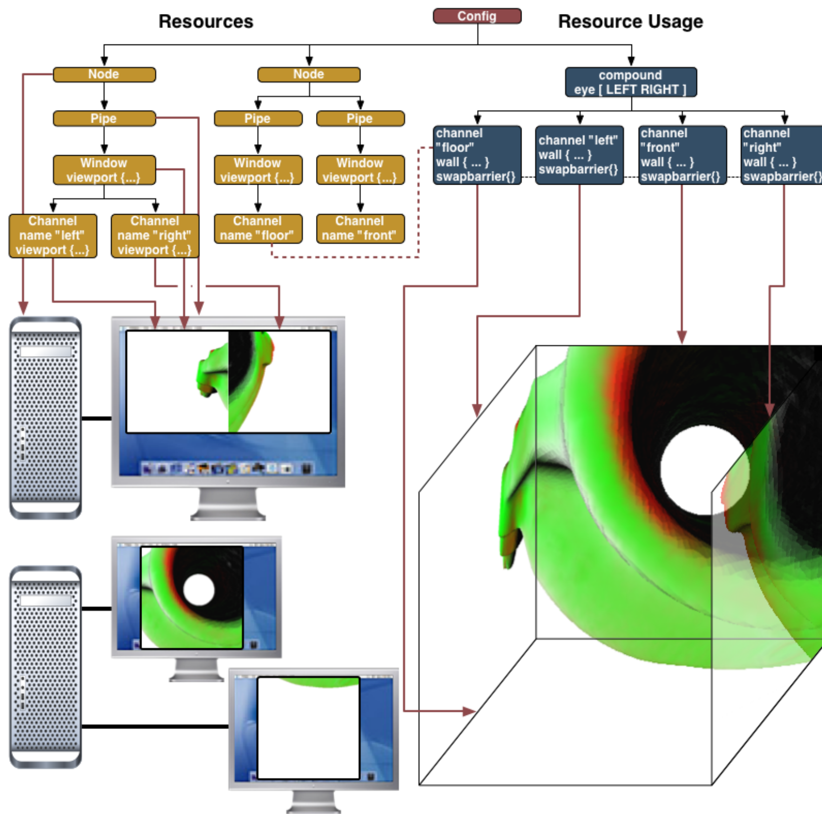
# Equalizer concepts and classes (2)




Equalizer sample configuration (source: Equalizer web site)

- Window
  - Holds drawable and OpenGL context

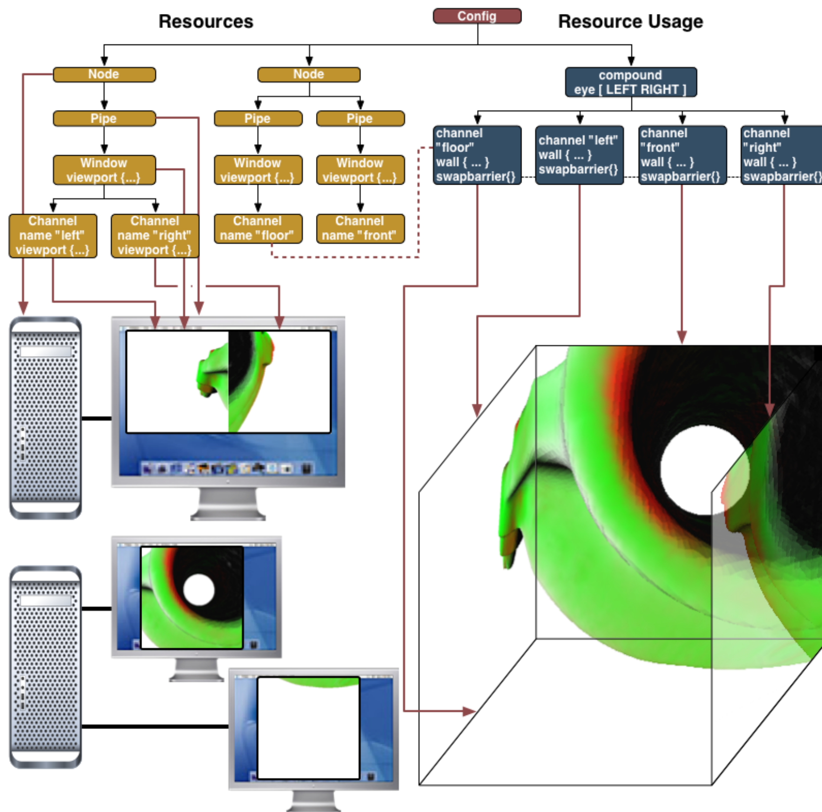
# Equalizer concepts and classes (2)




Equalizer sample configuration (source: Equalizer web site)

- Window
  - Holds drawable and OpenGL context
- Channel
  - Abstraction of viewport within the parent window
  - Entity responsible for the main rendering work

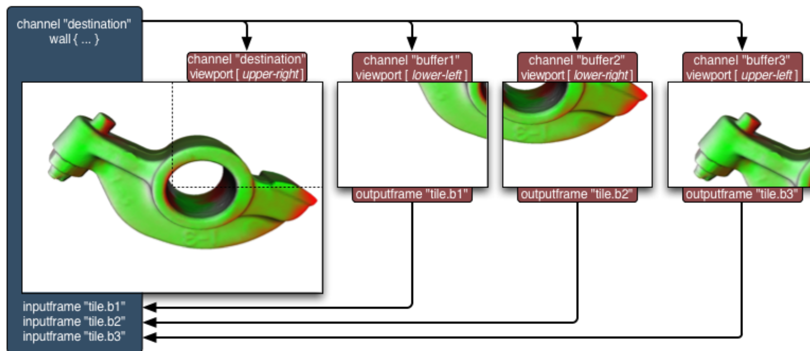
# Equalizer concepts and classes (2)



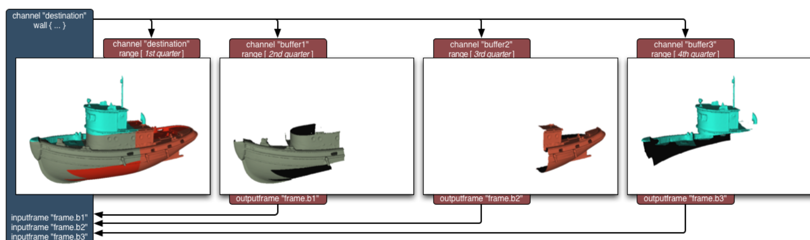

Equalizer sample configuration (source: Equalizer web site)

- Window
  - Holds drawable and OpenGL context
- Channel
  - Abstraction of viewport within the parent window
  - Entity responsible for the main rendering work
- Distributed objects
  - Static (unversioned)
  - Dynamic (versioned)

# Equalizer decomposition modes




Equalizer 2D decomposition (source: Equalizer web site)



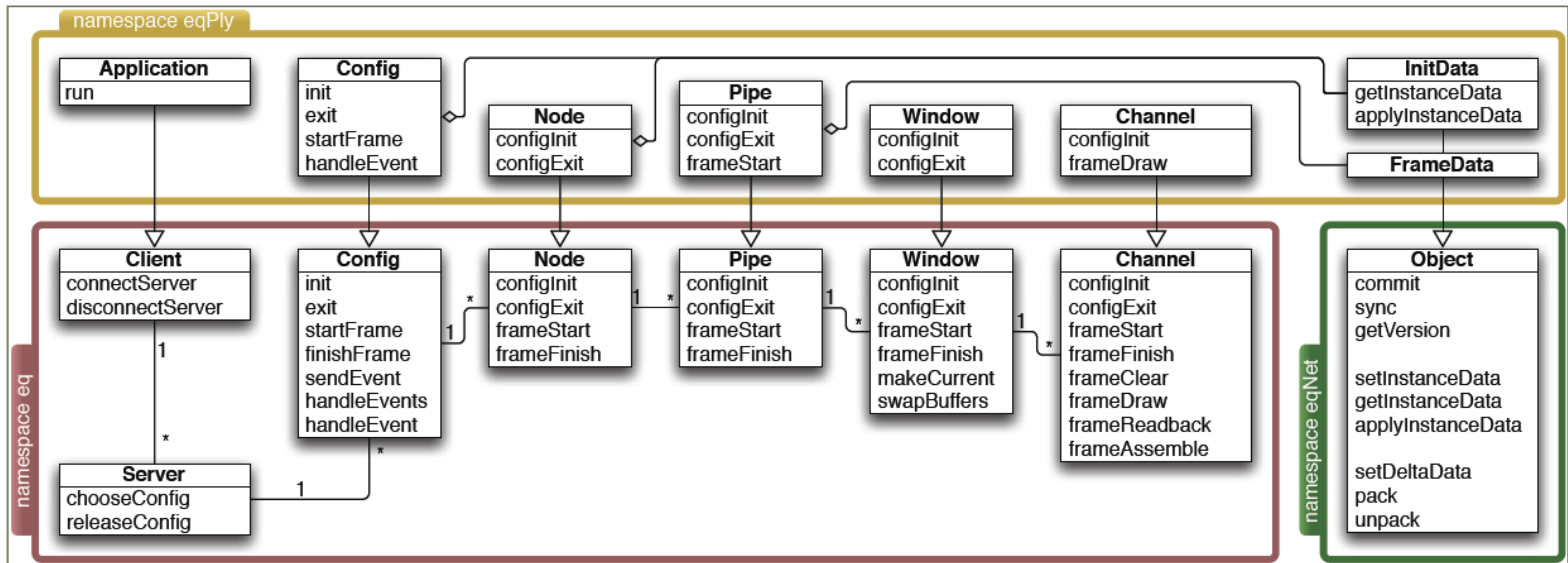

Equalizer DB decomposition (source: Equalizer web site)

- 2D
  - Partitions 2D screen space into multiple tiles
  - Relies on view frustum culling
- DB
  - Partitions data structure
  - Requires range support
- Eye
  - Partitions eye passes
  - Used for stereo rendering

# Outline

- Introduction
- **Application Design**
  - Equalizer
  - **eqPly**
- Data Structure
  - Old
  - New
- Rendering
- Summary

# eqPly overview




UML diagram of important Equalizer and eqPly classes
(source: Equalizer Programming Guide)

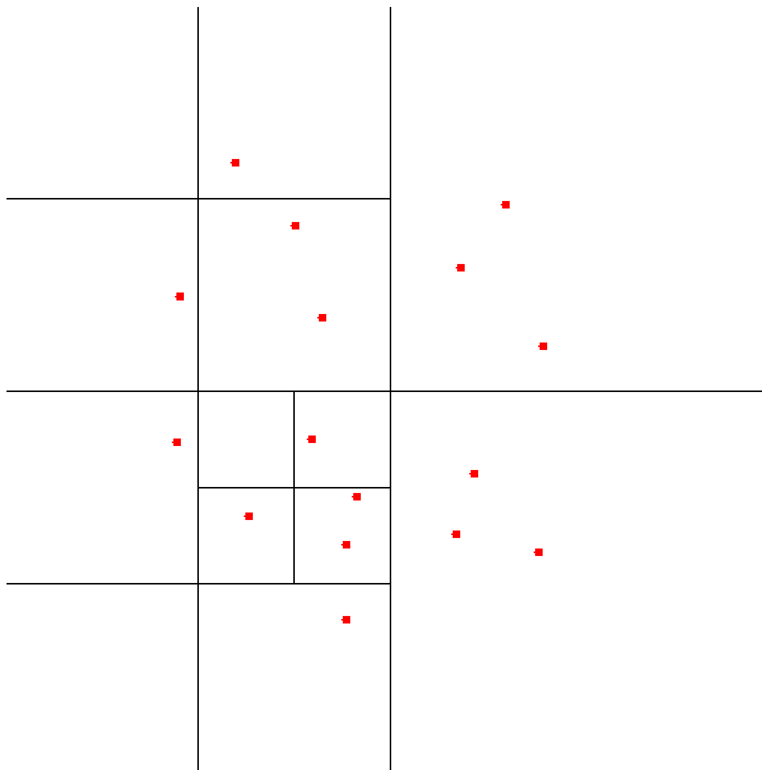
# Outline

- Introduction
- Application Design
  - Equalizer
  - eqPly
- **Data Structure**
  - **Old**
  - New
- Rendering
- Summary

# Old data structure (1)

- Bounding box octree
- Recursive algorithm
  - Nodes are filled with data until a certain threshold is reached
  - On overflow space is partitioned through the midpoint into equal sized octants
  - Recursion stops when data size in leaf stays below the given threshold, or a predefined maximum tree depth has been reached
- Implementation
  - Heavily template based
  - Structs for Vertex type, Face type and BBox node type

# Old data structure (2)



2D space partitioning with a quadtree

- Algorithm problems
  - Space partitioned evenly without regard to actual data distribution
  - Little control over resulting leaf sizes or overall tree balance
- Implementation problems
  - Large memory footprint
  - Potential memory leaks
  - Loss of structural information
  - Restriction to flat shading
  - Legacy C based code

# Outline

- **Introduction**
- Application Design
  - Equalizer
  - eqPly
- **Data Structure**
  - Old
  - **New**
- Rendering
- Summary

# New data structure (1)

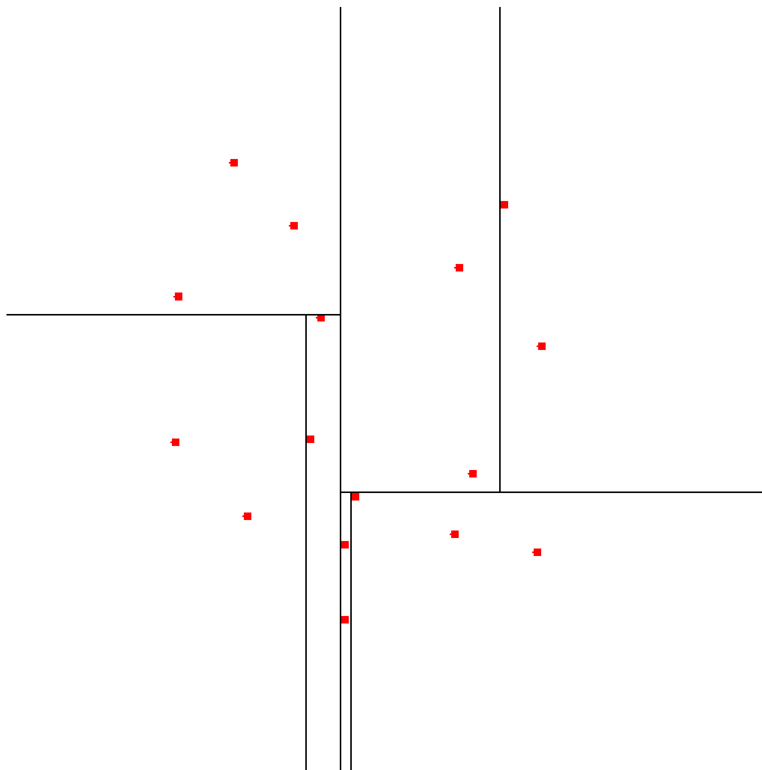
- Requirements
  - Avert the problems found in the old data structure
  - Keep structural information, support smooth shading
  - Optimize for Vertex Buffer Object (VBO) usage
  - Use contemporary object-oriented design and C++ features

# New data structure (1)

- Requirements
  - Avert the problems found in the old data structure
  - Keep structural information, support smooth shading
  - Optimize for Vertex Buffer Object (VBO) usage
  - Use contemporary object-oriented design and C++ features

- kd-tree
  - Sort data along one axis, rotated every recursion step
  - Determine median in regard to the chosen axis
  - Partition data into two halves, elements < median and elements ≥ median
  - Continue recursion until leaf size remains below wanted threshold

# New data structure (2)



2D space partitioning with a kd-tree

- Direct advantages
  - Smaller number of total nodes
  - (Nearly) equal data distribution
  - Well-balanced tree
- Remaining requirements
  - Preservation of the structural information
  - …

→ Design and implementation

# New design (1)

```
class VertexData
{
public:
    VertexData();

    bool readPlyFile( const char* file, const bool ignoreColors );
    void sort( const Index start, const Index length, const Axis axis );
    void scale( const float baseSize );
    void calculateNormals( const bool vertexNormals );
    void calculateBoundingBox();
    const BoundingBox& getBoundingBox() const;

    std::vector< Vertex >   vertices;
    std::vector< Color >    colors;
    std::vector< Normal >   normals;
    std::vector< Triangle > triangles;

private:
    ...
    BoundingBox _boundingBox;
};
```

# New design (2)

```
class VertexBufferData
{
public:
    void clear();
    void toStream( std::ostream& os );
    void fromMemory( char** addr );

    std::vector< Vertex >     vertices;
    std::vector< Color >      colors;
    std::vector< Normal >     normals;
    std::vector< ShortIndex > indices;

private:
    ...
};
```

```
class VertexBufferState
{
public:
    virtual bool useColors() const;
    virtual void setColors( const bool colors );
    virtual RenderMode getRenderMode() const;
    virtual void setRenderMode( const RenderMode mode );

    virtual GLuint getDisplayList( const void* key );
    virtual GLuint newDisplayList( const void* key );
    virtual GLuint getBufferObject( const void* key );
    virtual GLuint newBufferObject( const void* key );
    virtual void deleteAll();

    virtual const GLFunctions* getGLFunctions() const;

protected:
    VertexBufferState( const GLFunctions* glFunctions );
    virtual ~VertexBufferState();

    const GLFunctions*   _glFunctions;
    bool                 _useColors;
    RenderMode           _renderMode;
};
```

# New design (3)

```
class VertexBufferBase
{
public:
    virtual void render( VertexBufferState& state ) const;
    const BoundingSphere& getBoundingSphere() const;
    const float* getRange() const;
    virtual const VertexBufferBase* getLeft() const;
    virtual const VertexBufferBase* getRight() const;

protected:
    VertexBufferBase();
    virtual ~VertexBufferBase();

    virtual void toStream( std::ostream& os );
    virtual void fromMemory( char** addr, VertexBufferData& globalData );

    virtual void setupTree( VertexData& data, const Index start, const Index length, const
                            Axis axis, const size_t depth, VertexBufferData& globalData );
    virtual BoundingBox updateBoundingSphere();
    virtual void updateRange();
    void calculateBoundingSphere( const BoundingBox& bBox );

    BoundingSphere  _boundingSphere;
    Range           _range;
};
```

# New design (4)

```
class VertexBufferNode : public VertexBufferBase
{
    ...
private:
    size_t countUniqueVertices( VertexData& data, const Index start, const Index length ) const;

    VertexBufferBase*   _left;
    VertexBufferBase*   _right;
};



class VertexBufferLeaf : public VertexBufferBase
{
    ...
private:
    ...

    VertexBufferData&   _globalData;
    Index               _vertexStart;
    ShortIndex          _vertexLength;
    Index               _indexStart;
    Index               _indexLength;
};
```

# New design (5)

```
class VertexBufferRoot : public VertexBufferNode
{
public:
    virtual void render( VertexBufferState& state ) const;

    void beginRendering( VertexBufferState& state ) const;
    void endRendering( VertexBufferState& state ) const;

    void setupTree( VertexData& data );
    bool writeToFile( const char* filename );
    bool readFromFile( const char* filename );
    bool hasColors() const;

protected:
    virtual void toStream( std::ostream& os );
    virtual void fromMemory( char* start );

private:
    bool constructFromPly( const char* filename );

    VertexBufferData _data;
};
```

# New implementation (1)

```
void VertexBufferNode::setupTree( VertexData& data, const Index start,
                                  const Index length, const Axis axis,
                                  const size_t depth,
                                  VertexBufferData& globalData )
{
    data.sort( start, length, axis );
    const Index median = start + ( length / 2 );

    // left child will include elements smaller than the median
    const Index leftLength    = length / 2;
    const bool  subdivideLeft =
        countUniqueVertices( data, start, leftLength ) > LEAF_SIZE || depth < 3;

    if( subdivideLeft )
        _left = new VertexBufferNode;
    else
        _left = new VertexBufferLeaf( globalData );

    ...
```

# New implementation (2)

```
    ...

    // right child will include elements equal or greater median
    const Index rightLength    = ( length + 1 ) / 2;
    const bool  subdivideRight =
        countUniqueVertices( data, median, rightLength ) > LEAF_SIZE || depth < 3;

    if( subdivideRight )
        _right = new VertexBufferNode;
    else
        _right = new VertexBufferLeaf( globalData );

    // move to next axis and continue contruction in the child nodes
    const Axis newAxis = static_cast< Axis >( ( axis + 1 ) % 3 );
    static_cast< VertexBufferNode* >
        ( _left )->setupTree( data, start, leftLength, newAxis,
                              depth + 1, globalData );
    static_cast< VertexBufferNode* >
        ( _right )->setupTree( data, median, rightLength, newAxis,
                               depth + 1, globalData );
}
```

# New implementation (3)

```
void VertexBufferLeaf::setupTree( VertexData& data, const Index start,
                                  const Index length, const Axis axis,
                                  const size_t depth,
                                  VertexBufferData& globalData )
{
    data.sort( start, length, axis );
    _vertexStart  = globalData.vertices.size();
    _vertexLength = 0;
    _indexStart   = globalData.indices.size();
    _indexLength  = 0;

    const bool hasColors = ( data.colors.size() > 0 );

    // stores the new indices (relative to _start)
    map< Index, ShortIndex > newIndex;

    ...
```

# New implementation (4)

```
    ...

    for( Index t = 0; t < length; ++t )
    {
        for( Index v = 0; v < 3; ++v )
        {
            Index i = data.triangles[start + t][v];
            if( newIndex.find( i ) == newIndex.end() )
            {
                newIndex[i] = _vertexLength++;
                // assert number does not exceed SmallIndex range
                MESHASSERT( _vertexLength );
                globalData.vertices.push_back( data.vertices[i] );
                if( hasColors )
                    globalData.colors.push_back( data.colors[i] );
                globalData.normals.push_back( data.normals[i] );
            }
            globalData.indices.push_back( newIndex[i] );
            ++_indexLength;
        }
    }
}
```

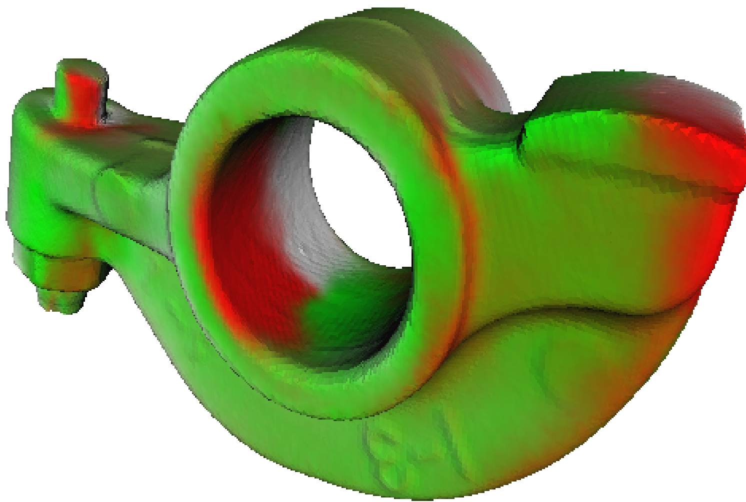
# Outline

- Introduction
- Application Design
  - Equalizer
  - eqPly
- Data Structure
  - Old
  - New
- **Rendering**
- Summary

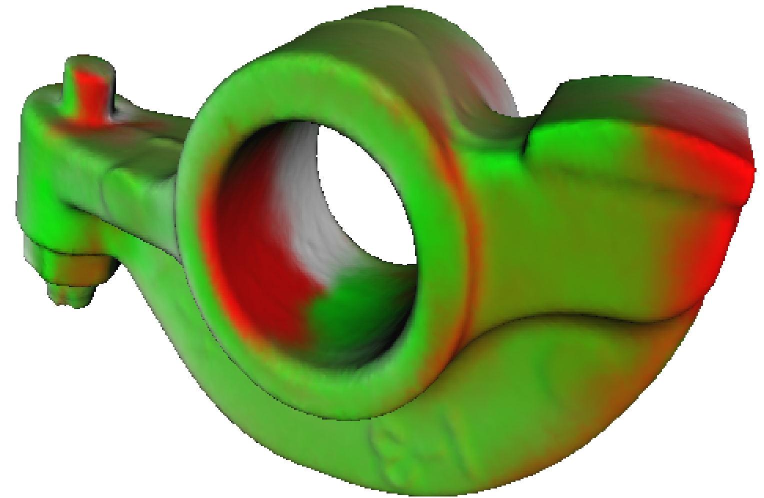
# Rendering

- Vertex Buffer Objects
  - Originally an extension, since OpenGL 1.5 core functionality
  - “Vertex arrays on steroids”, stored directly on the GPU
  - Static and dynamic data supported
- Performance considerations
  - Multiple function calls
  - Choice of batch size ,data types
- Shading algorithms
  - Flat shading
  - Gouraud shading
  - Phong shading

# Flat vs. smooth shading

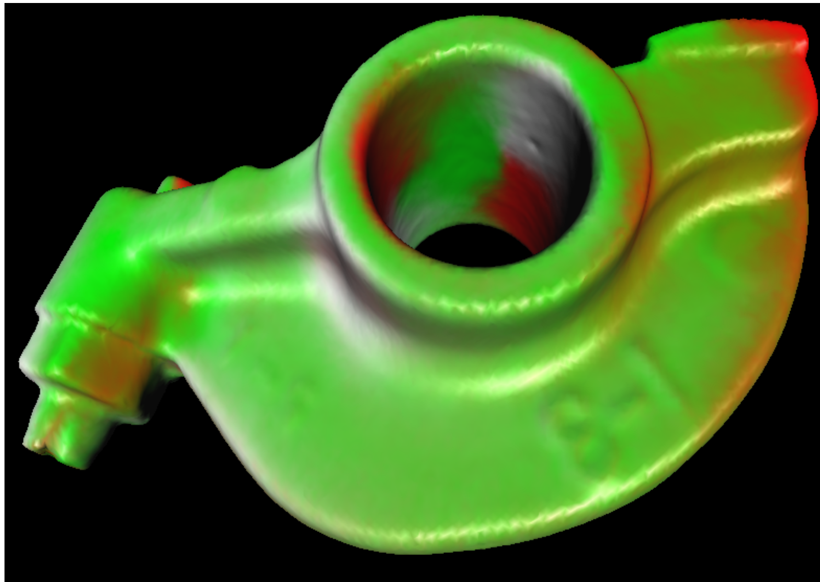


RockerArm model with original flat shading

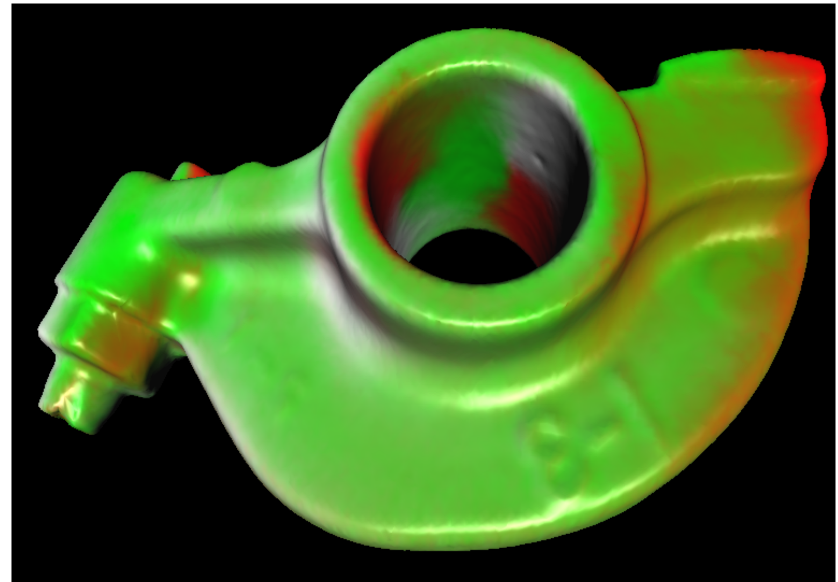


RockerArm model with new smooth shading

# Gouraud vs. Phong shading



Shiny RockerArm model with Gouraud shading



Shiny RockerArm model with Phong shading

# Outline

- Introduction
- Application Design
  - Equalizer
  - eqPly
- Data Structure
  - Old
  - New
- Rendering
- **Summary**

# Summary (1)

- First goal fully achieved
  - Only drawback is the slow tree setup time
  - Alleviated by the use of binary representations

| Model | Binary octree [MB] | Binary kd-tree [MB] | Ratio [%] |
|---|---:|---:|---:|
| Armadillo | 27.72 | 6.09 | 21.96 |
| Dragon | 69.83 | 15.34 | 21.97 |
| Happy Buddha | 87.16 | 19.05 | 21.86 |
| Lucy | n/a | 494.58 | n/a |
| Stanford Bunny | 5.56 | 1.20 | 21.56 |

File size differences between the binary representations

# Summary (2)

- Second goal partially achieved
  - Vertex Buffer Object rendering
  - OpenGL 1.1 display list fallback
  - Overall better performance
  - GLSL shader support
  - Gouraud and Phong shading

- But
  - VBO performance difference varying wildly
  - Out of time to implement further shader effects

# Summary (3)

| Model | Number of triangles | Number of leaves | Triangles per leaf | Vertices per leaf | Old DL [fps] | New DL [fps] | New VBO [fps] |
|---|---|---|---|---|---|---|---|
| Stanford Bunny | 69’451 | 16 | 4’340 | 2’329 | 406.82 | 436.67 | 617.00 |
| Rocker Arm | 80’354 | 16 | 5’022 | 2’797 | 445.61 | 445.57 | 301.67 |
| Goddess 1 | 274’822 | 16 | 17’176 | 9’252 | 142.20 | 162.45 | 70.94 |
| Armadillo | 345’944 | 16 | 21’621 | 11’218 | 118.30 | 136.43 | 92.17 |
| Boat | 776’374 | 32 | 24’261 | 12’769 | 29.25 | 20.06 | 28.47 |
| Dragon | 871’414 | 32 | 27’231 | 14’138 | 17.06 | 45.02 | 38.19 |
| Goddess 2 | 1’047’330 | 32 | 32’729 | 17’430 | 10.02 | 9.32 | 42.99 |
| Hip | 1’060’346 | 32 | 33’135 | 17’307 | 11.10 | 9.24 | 29.37 |
| Happy Buddha | 1’087’716 | 32 | 33’991 | 17’509 | 9.54 | 29.27 | 34.88 |

Performance and statistical data for a few selected models

Questions or comments?

Semester Thesis:

# Development of a Parallel OpenGL Polygon Renderer

Tobias Wolf

19. December 2007