3D Viewers In ROOT

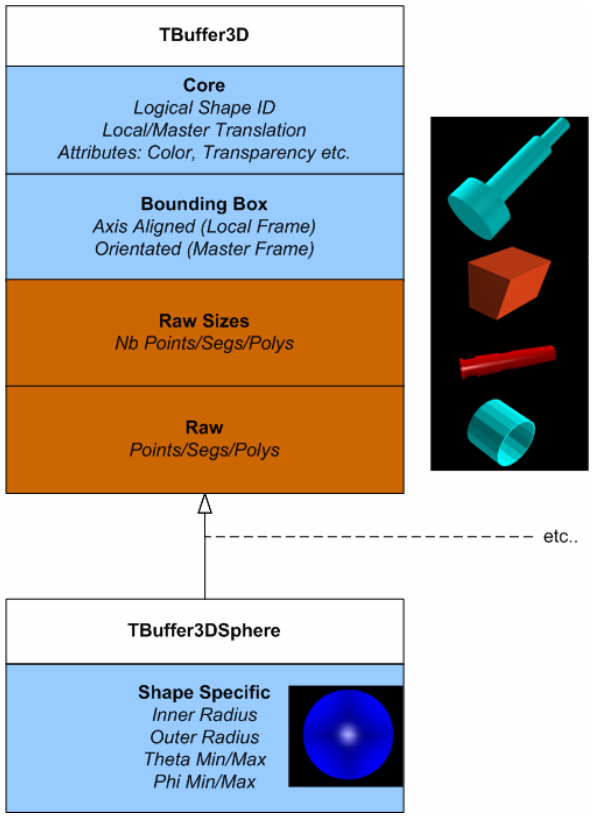
Richard Maunder / Timur Pocheptsov

Overview

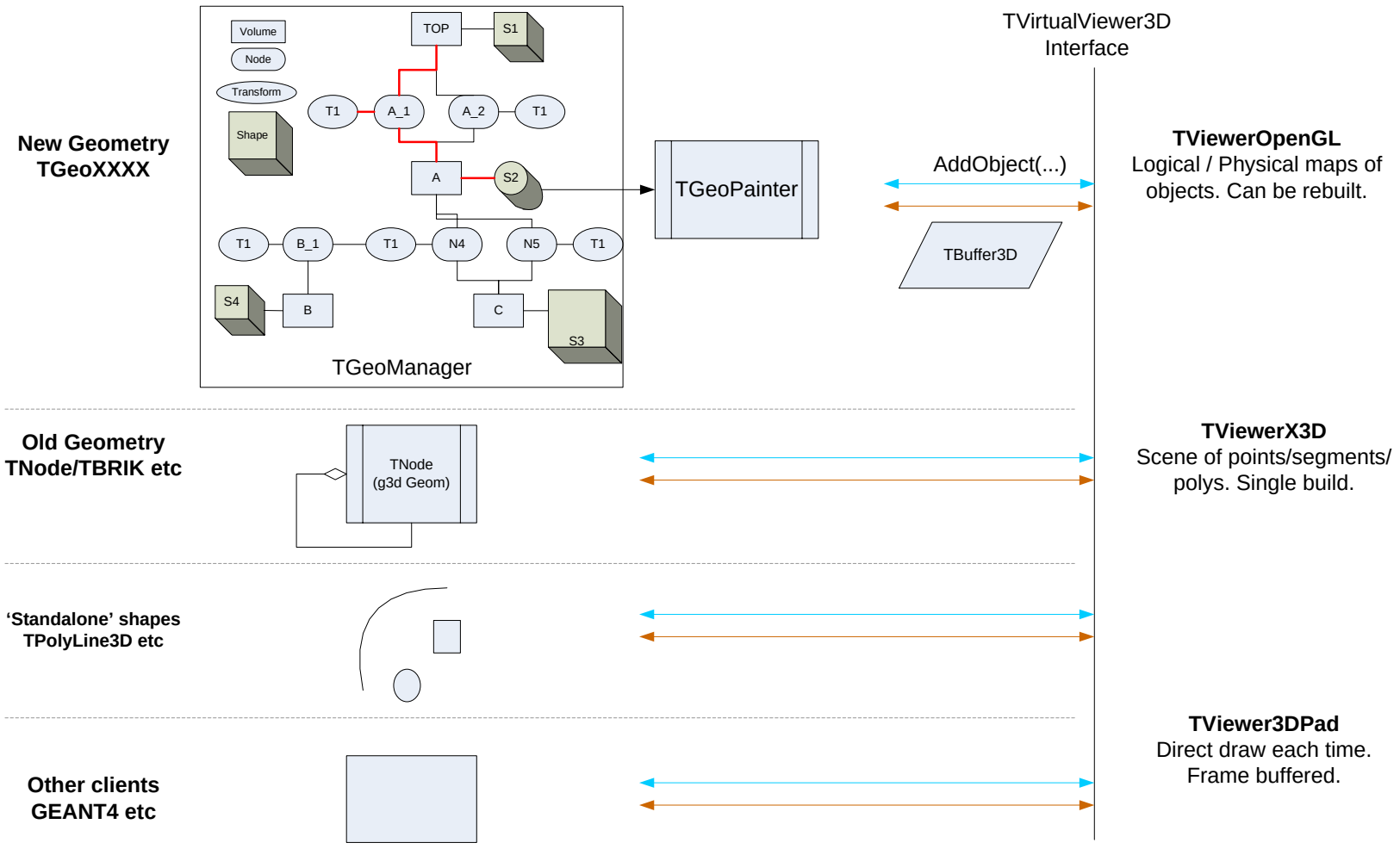
- General Viewer Architecture
- GL Architecture
- GL Features
- GL Performance
- Conclusion

General Architecture I

- All current and future 3D viewers share common external infrastructure:
  - **TVirtualViewer3D** interface:
    - Test viewer preferences and capabilities.
    - Adding objects – including composite operations.
    - Control the viewer via scripting etc.
  - **TBuffer3D** class hierarchy:
    - Describe 3D objects ("shapes").
    - Split into sections, filled by negotiation with viewer.
    - Base TBuffer3D for common sections
    - TBuffer3D sufficient for any object in raw tessellated* form
    - Subclasses for abstract shapes which viewer(s) can tessellate natively
- These enable:
  - Decoupling of producers (geometry packages etc) who model collection of 3D objects from consumers (viewers) which display.
  - Producer free of explicit drawing code & viewer specific branching.
  - Support differing viewers and clients efficiently:
    - Local/global frame
    - Bounding boxes
    - Individual objects / placed copies



General Architecture II

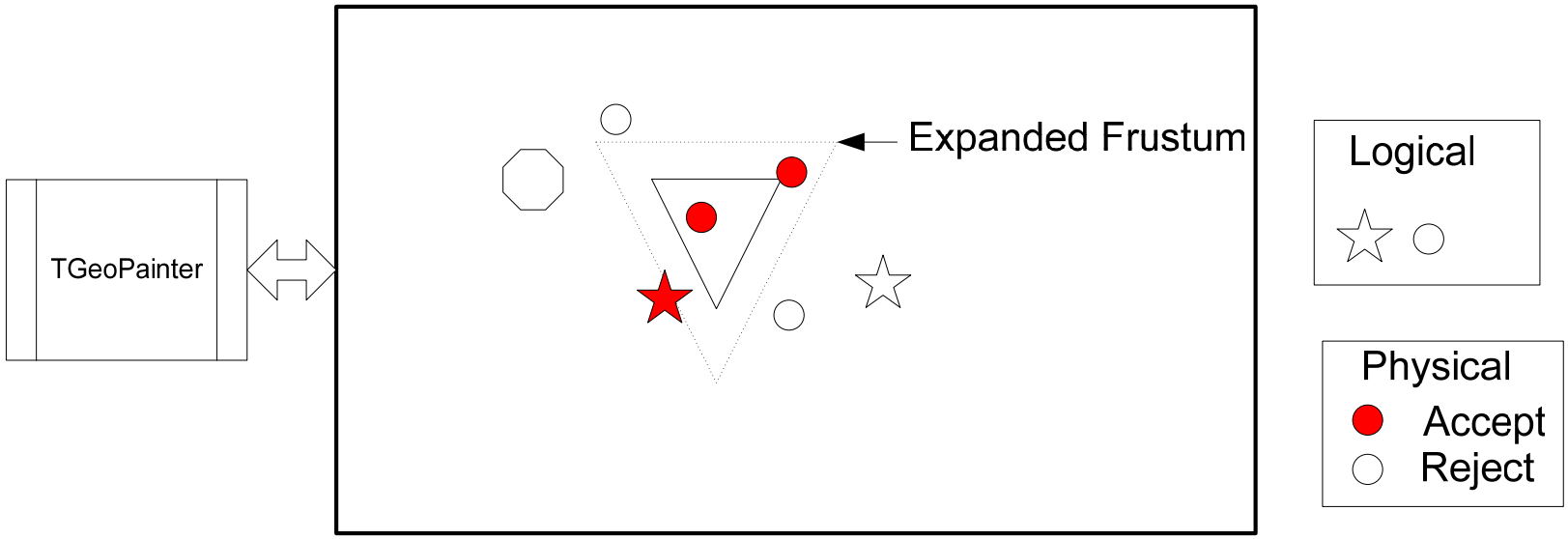


GL Architecture

- X3D and Pad simple viewers:
  - Accept all objects send – with no caching or detection of copied placements.
  - No viewer side tessellation – always request Raw section of TBuffer3D to be filled.
- By contrast GL:
  - Only accept subset of objects considered 'of interest' at present into viewer.
  - Viewer can prompt external client to rebuild scene of objects when camera limits have changed significantly.
  - Can request client not to send contained children of object via TVV3D::AddObject() return – for efficiency.
  - Enables viewer to connect to very large geometries without overloading - pull required parts on demand.
- Detect repeated placement of same object (logical) in different 3D locations (physical).
- Lazy caching of logicals (with expensive internal or externally created tessellation) and physicals.
- Native OpenGL shapes:
  - TBuffer3DSphere - solid, hollow and cut spheres*
  - TBuffer3DTubeSeg - angle tube segment
  - TBuffer3DCutTube - angle tube segment with plane cut ends.

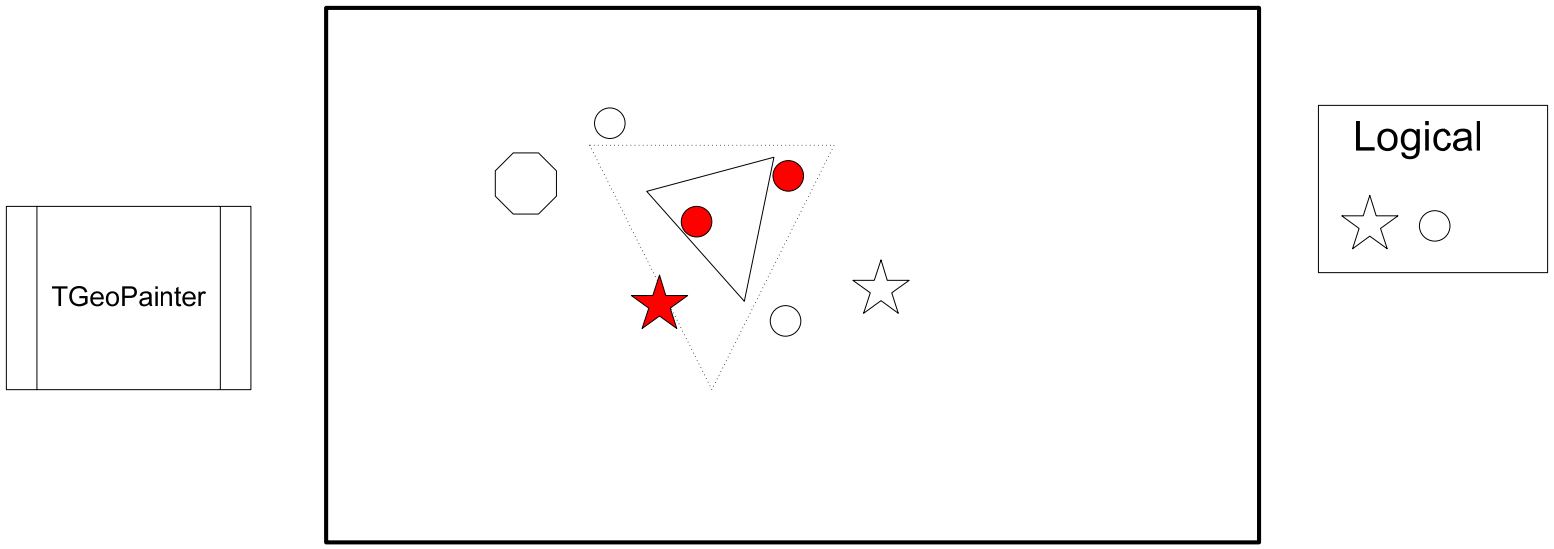
- Scene rebuild through binding to pad –TPad::Paint. Remove – make all communication with external client via signals (publish scene, selection change)

GL Architecture : Scene Rebuilds I



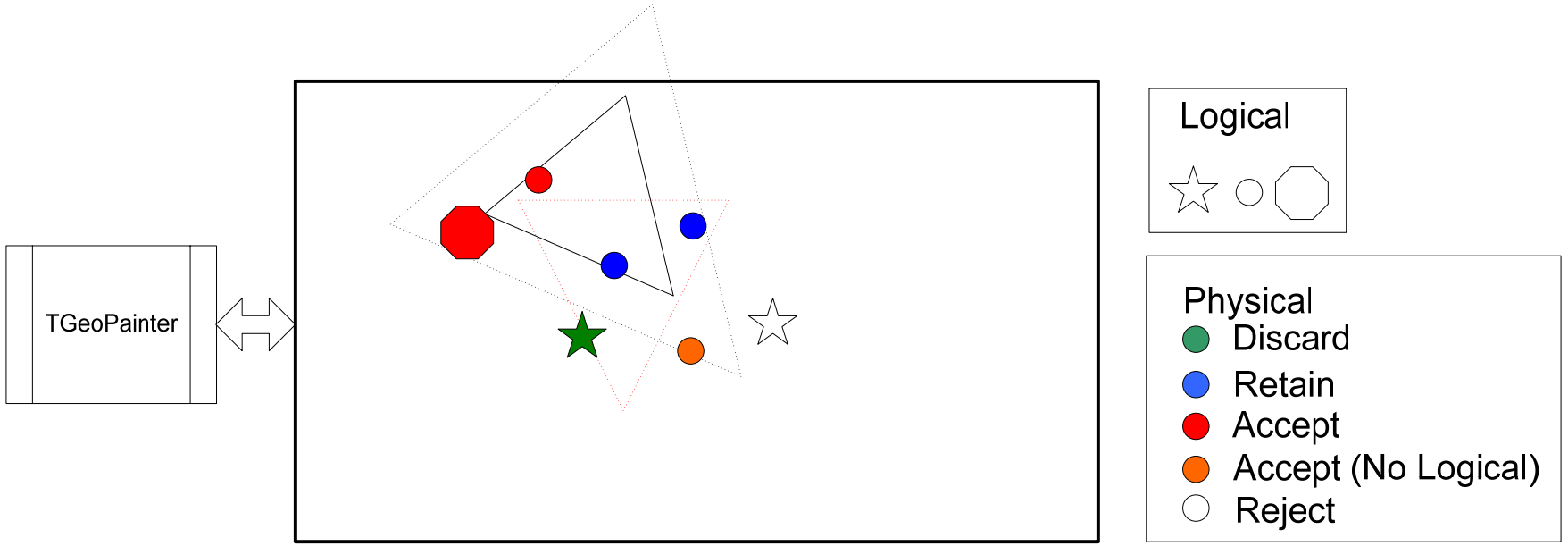
Accept objects (and children) falling inside Expanded Frustum.
Terminate branch expansion when object too small.

GL Architecture : Scene Rebuilds II



Small perturbations in camera require no new geometry pull

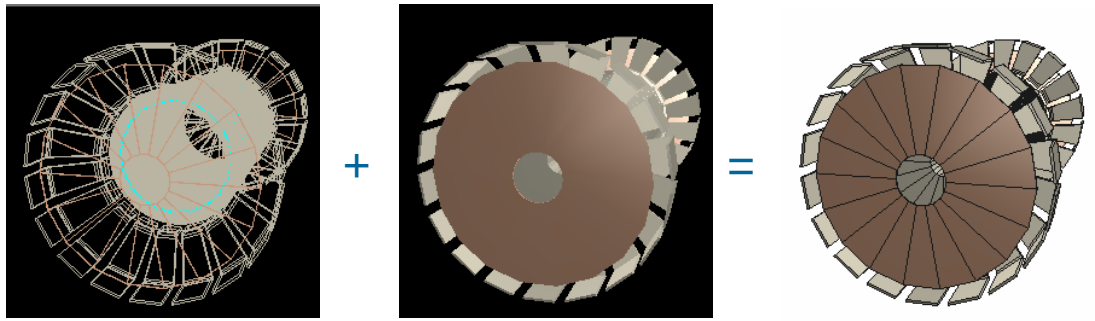
GL Architecture : Scene Rebuilds III



Large perturbation – new geometry pull. Performed at end of camera move.
Physical shapes inside new expanded frustum retained.
Physical shapes outside expanded frustum discarded/recycled.
All Logical Shapes retained – even if all referencing physicals discarded.
Logical with no physical ref is candidate for mesh/DL purge.

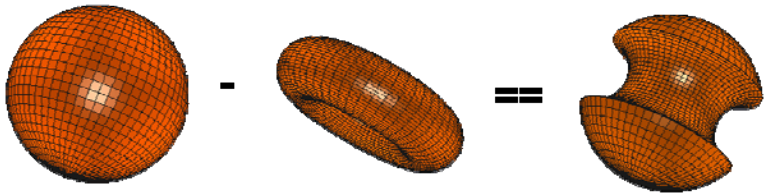
GL Features : Rendering

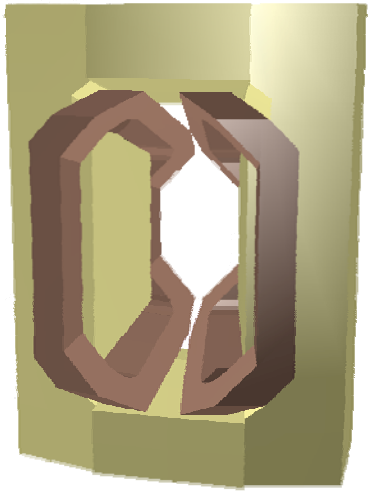
- 3 draw styles



Wireframe Filled polygons Outline

- Added support for single frame pdf and eps output.
- Composite (CSG) shapes





- Both TGeoComposite and general cases via TVirtualViewer3D interface:

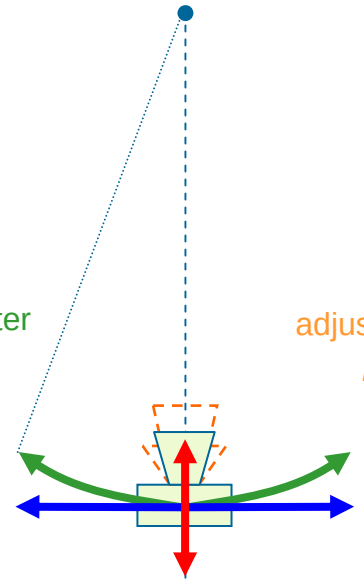
```
virtual Bool_t OpenComposite(const TBuffer3D & buffer, Bool_t * addChildren = 0) = 0;
virtual void CloseComposite() = 0;
virtual void AddCompositeOp(UInt_t operation) = 0;
```

kCSUnion, kCSIntersection, kCSDifference, kCSNoOp

- Not supported in X3D/Pad viewers – each composite component treated as individual object.

GL Features : Cameras

- Improved interactions



Orbit
rotate round scene center
Left Mouse + Drag

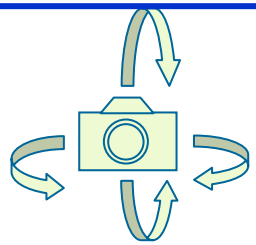
Zoom
adjust lens field of view
Mouse Wheel
j / k keys

Dolly
move camera in/out along eye line
Right Mouse + Horizontal Drag

Truck
pan parallel to film plane
Middle Mouse + Drag
arrow keys

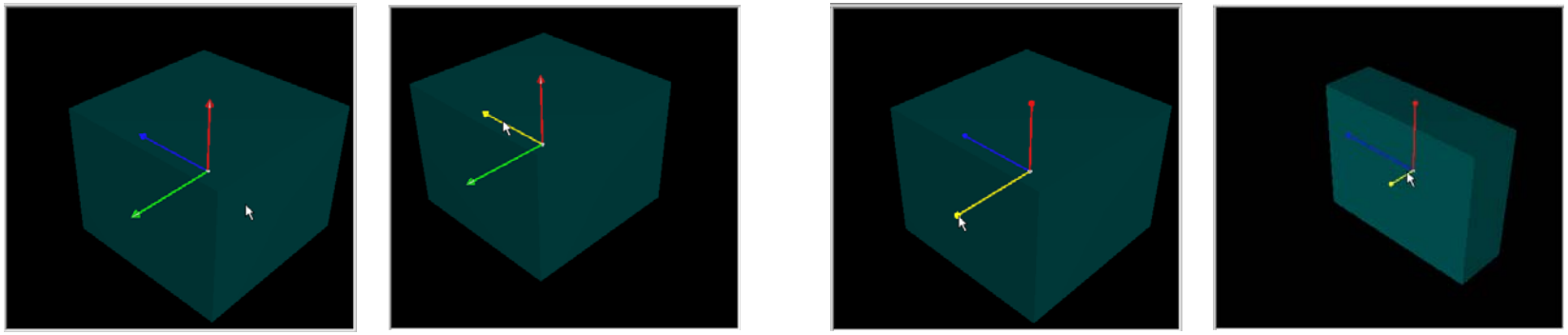
- Shift: x 10
- Ctrl: x 0.1
- Shift+Ctrl: x 0.01
- Double click reset

- 'Heads up' camera – first person shooting games.
- Orbit round own axis with mouse – truck with arrow keys.
- Box zoom, frame all/selected.



GL Features : Manipulators

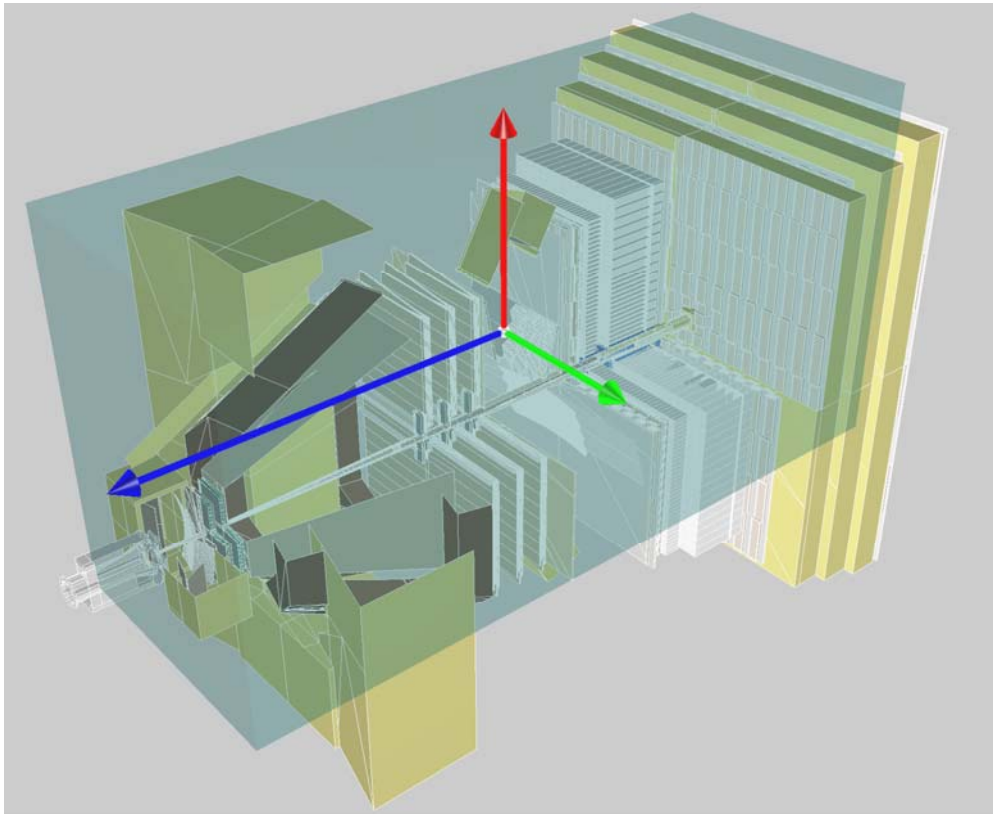
- Add in-viewer manipulators for direct control of object
  - Translation & scaling along objects local axis.

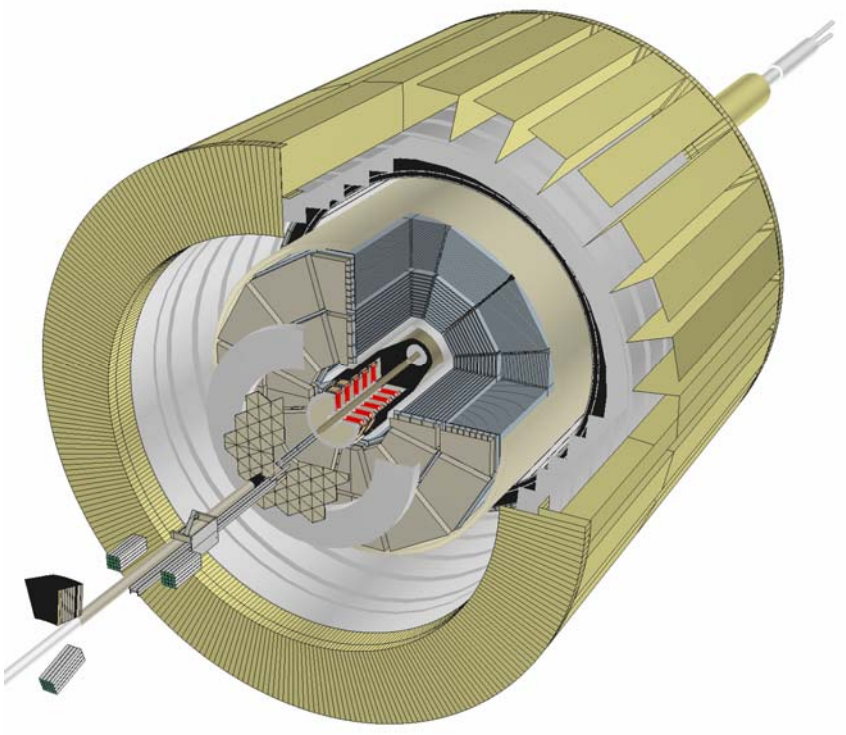


  - Rotation of these local axis.

- Implemented for clipping objects – trivial to extend for all objects.

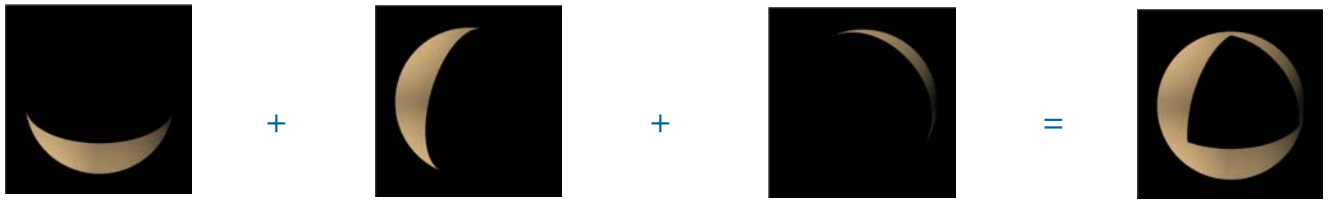
GL Features : Clipping





GL Features : Clipping

- Two techniques
- OpenGL® Clip Planes: Multiple renders, each with one or more clip planes, combine together



+ Fast and simple, interactive (few planes)
- Accurate only for shapes described by planes – bounding box approx otherwise.
- Clipped solids not capped –hollow.

- CSG Operation: Add all object meshes (o1..on), subtract clipping object mesh (c)

$$o1 + o2 + ... + on - c$$

+ Any arbitrary clipping shape possible
+ Proper capping of solids
- Cannot adjust interactively

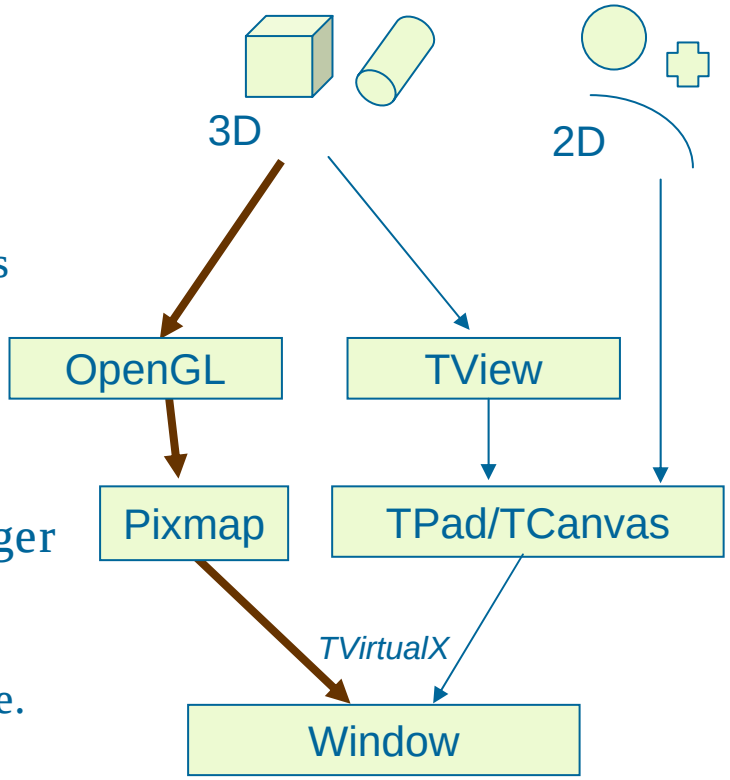
- Can clip away inside or outside the clip shape by negating planes or CSG operation.
- Separate 3D object collections with different clipping – e.g. detector geometry inside, events outside.

- Support both methods: clip planes for interactive setup (soon), CSG for high quality renders.

GL Features : GL – in - TPad

- Existing TPad/TCanvas has support for:
  - 2D via TVirtualX (X11/Win32 platform APIs)
  - 3D outlines via TView 3D -> 2D projection
  - 3D filled objects via ray tracing

- New method added:
  - Embedded OpenGL view, mixed with normal 2D graphics
  - 2D via TVirtualX (as now)
  - 3D via OpenGL
- Enable with gStyle->SetCanvasPreferGL(kTRUE)
- OpenGL is rendered into offscreen pixmap
- Bitmap and normal GL Window handled by TGLManager
- Advantages:
  - Gain all the features of the standalone GL viewer.
  - Only minor modifications in existing TPad \TCanvas code.
  - No need to duplicate 2D drawing in OpenGL.
- Disadvantages:
  - OpenGL may not be accelerated by graphics hardware – not ideal for complex geometries. But still faster than ray tracing.

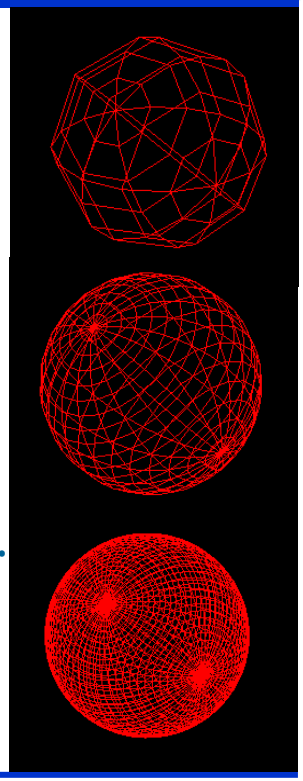


GL Performance : Culling

- Frustum Culling: Discard objects outside camera.
  - Test scene bounding first – if visible, all objects visible.
  - Otherwise test each shape BB - skip drawing those outside.
  - Significant 5-10x speed up (when viewing portion of scene)
- Contribution Culling: Discard small objects – part of LOD.
- Occlusion Culling: Object masked by others.
  - Potentially v useful given ‘onion’ layout of detector geometries.
  - But complicated - transparency, cut objects.
  - Left to OpenGL at present.

GL Performance : Object Draws

- Added generic support for:
- Level of Detail (LOD) scheme
  - Tessellate object at detail suitable for projected size + GL performance.
  - Distribute GPU power better: 5-10 x speed up for 'same quality'
  - Draw() methods take UInt_t LOD – quality factor 0 – 100%
  - LOD found from object bounding box projected onto screen
  - Combined with $Global_{LOD}$ to factor in overall GL performance.
- Display Lists
  - 'Pre-compile' draw command into efficient hardware specific GL format.
  - Added singleton DL cache – based on object ID and LOD.
  - Simple auto capture of any TGLDrawable by setting flags.
- Only TGLSphere (gluSphere) taking advantage at present – major



- Add more native shape types (common) supporting LOD + enable DL caching.
- Support high quality tessellation - degrade internally in response to LOD (various techniques)
- Add memory management in DisplayList cache – purge least frequent / oldest one to avoid thrashing.

GL Performance : Scene Draws

- Two-pass rendering of scene
- Interactive – speed:
  - Lower $Global_{LOD}$ – fixed 50% presently.
  - Skip very low $Draw_{LOD}$ (projected size) objects
  - Use sorted draw list: large -> small based on object bbox volume (true size).
  - Time limited – 100msec (10 fps) – rest discarded.
  - Ensure responsiveness with wide variety of hardware / software GL performance.
- Final – quality:
  - Unlimited time, everything drawn, $Global_{LOD}$ = 100%

- Interrupt/pause final pass render when GUI event enters queue – continue/terminate.
- Skip interactive if can complete final render in time slot.
- Make $Global_{LOD}$ settings dependent on performance.
- Split 'quality' pass into multiple accumulations – so low performance hardware can gradually 'fill in' details over extended (10 sec+) period, without stalling.
- Retain static draw buffers (geometry) – redraw varying (tracks/particles) over top.

Conclusion

- 3D Architecture:
  - Now stable – can extend easily for extra shapes etc
  - Add extensions for:
    - Scripted control of viewer via TVV3D - clipping, cameras, lights.
    - Efficient collections of particles/tracks.
    - Need user input on these.

- GL Viewer:
  - Internal structure now fairly stable + suitable for detector geometry.
  - Complete clipping and GL-in-Pad.
  - More native GL shapes to take advantage of performance/quality features.
  - General components for event display.