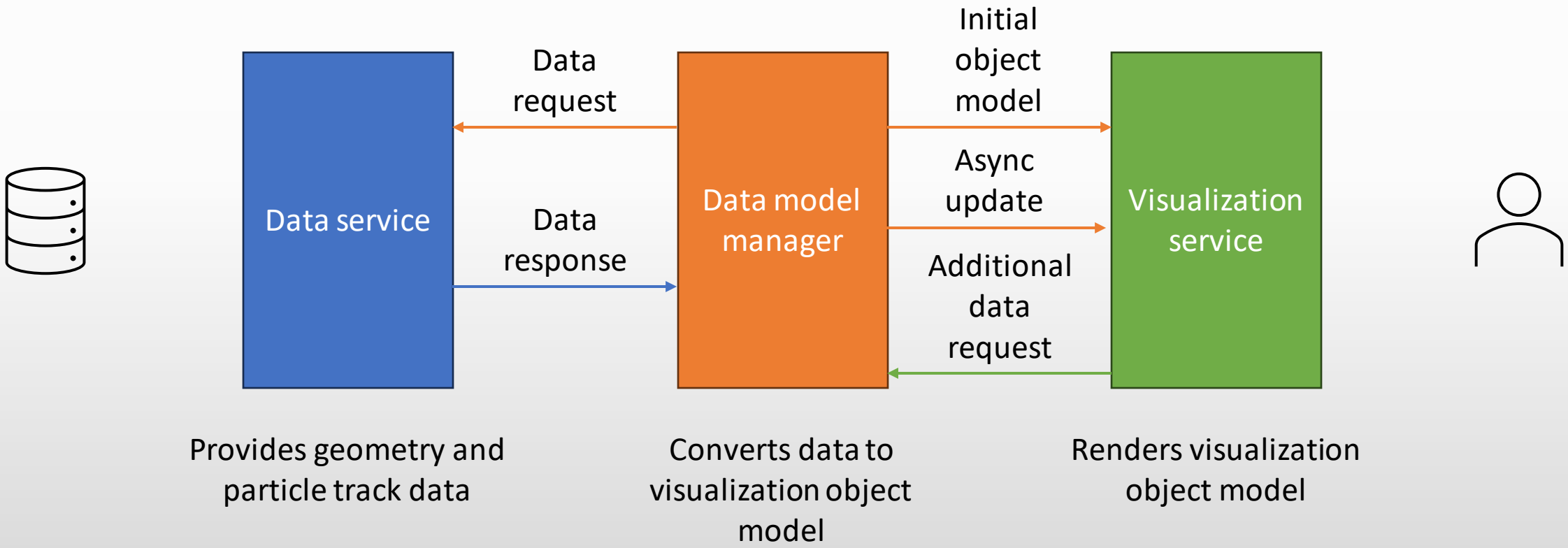
Development of Next-Gen Event Visualization Platform for BM@N

Previous work overview

- ROOT EVE - https://root.cern.ch/doc/master/group__TEve.html
- JSRoot - https://root.cern.ch/js/
- Phoenix - https://github.com/HSF/phoenix

Visualization platform structure



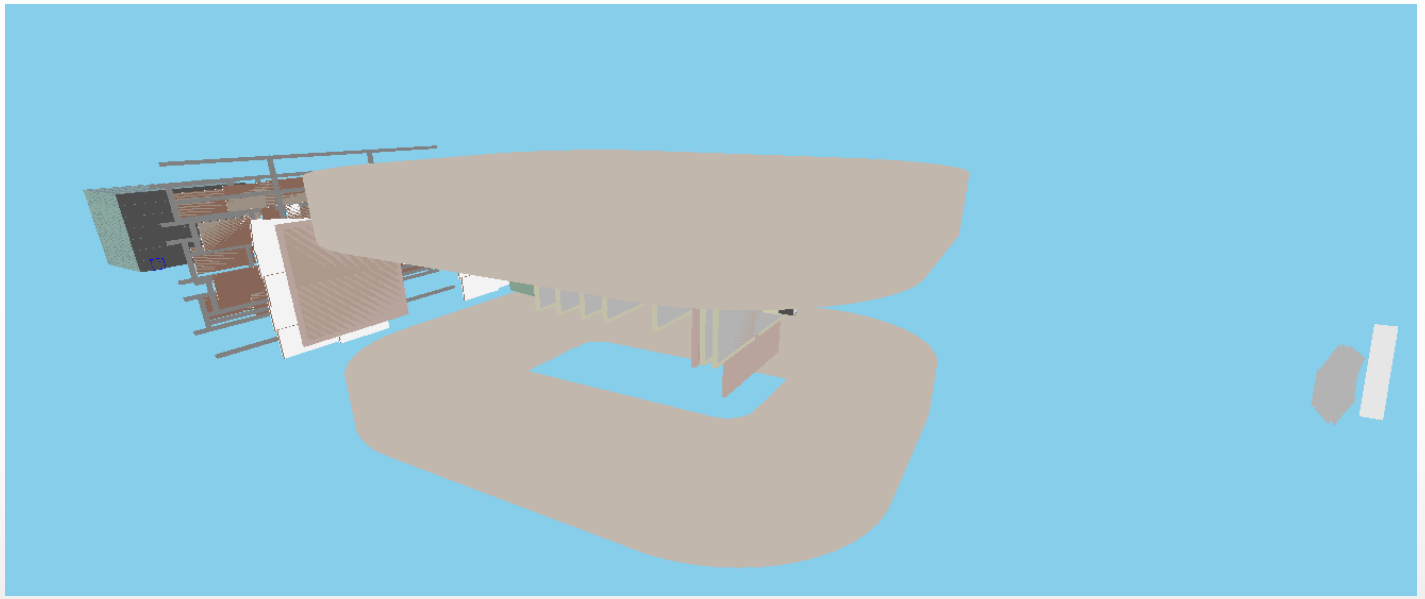
What is wrong with ROOT?

- Impractical object model
- Could be properly read only from ROOT itself
- Model is not observable (can re-render only the whole model, not its parts)

How to do better

- Create language-agnostic visualization object model.
- Make the model property-based so changes could be communicated as small patches.
- Support multiple renderers and multiple data sources.
- Multiplatform implementation.

VisionForge



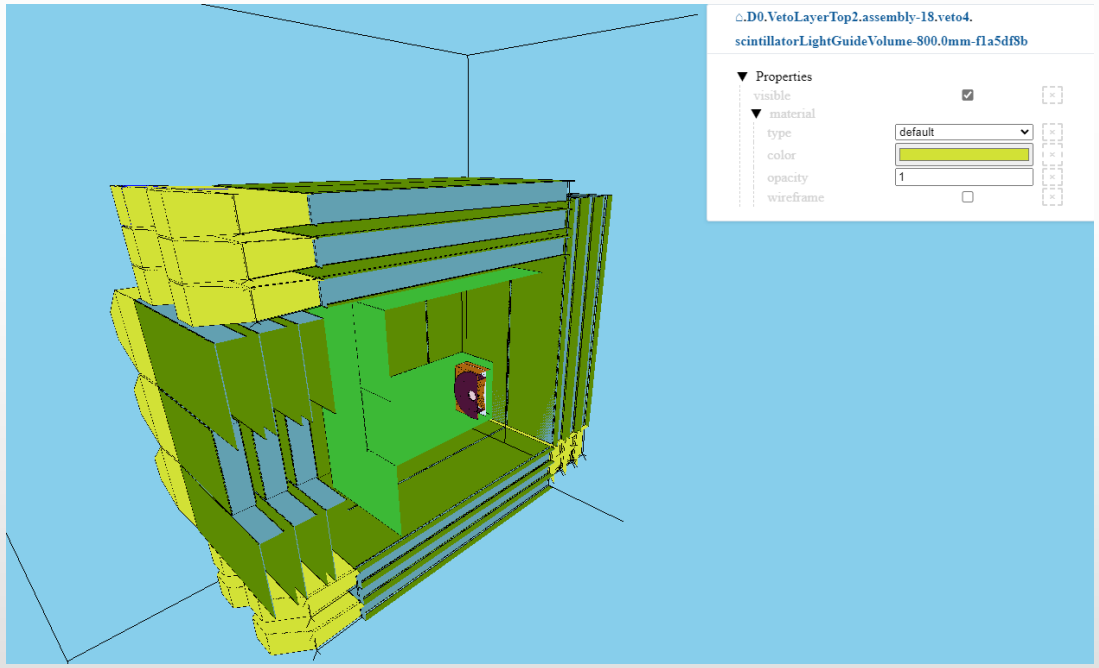
https://github.com/SciProgCentre/visionforge

Basic concepts

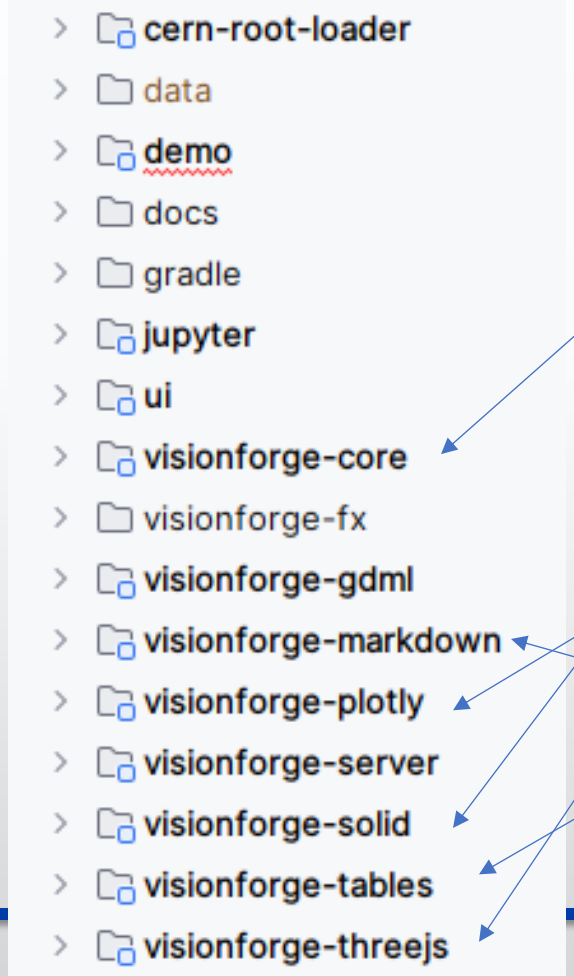
More details: https://youtu.be/uT5j-xOXC3E

- VisionForge is a visualization library written in Kotlin-Multiplatform.
- Core mechanics supports JVM, JS and Native builds (Wasm soon to be delivered).
- Primary rendering frontend for 3D is Three-JS (https://threejs.org/).

D0 geometry for Baby-IAXO



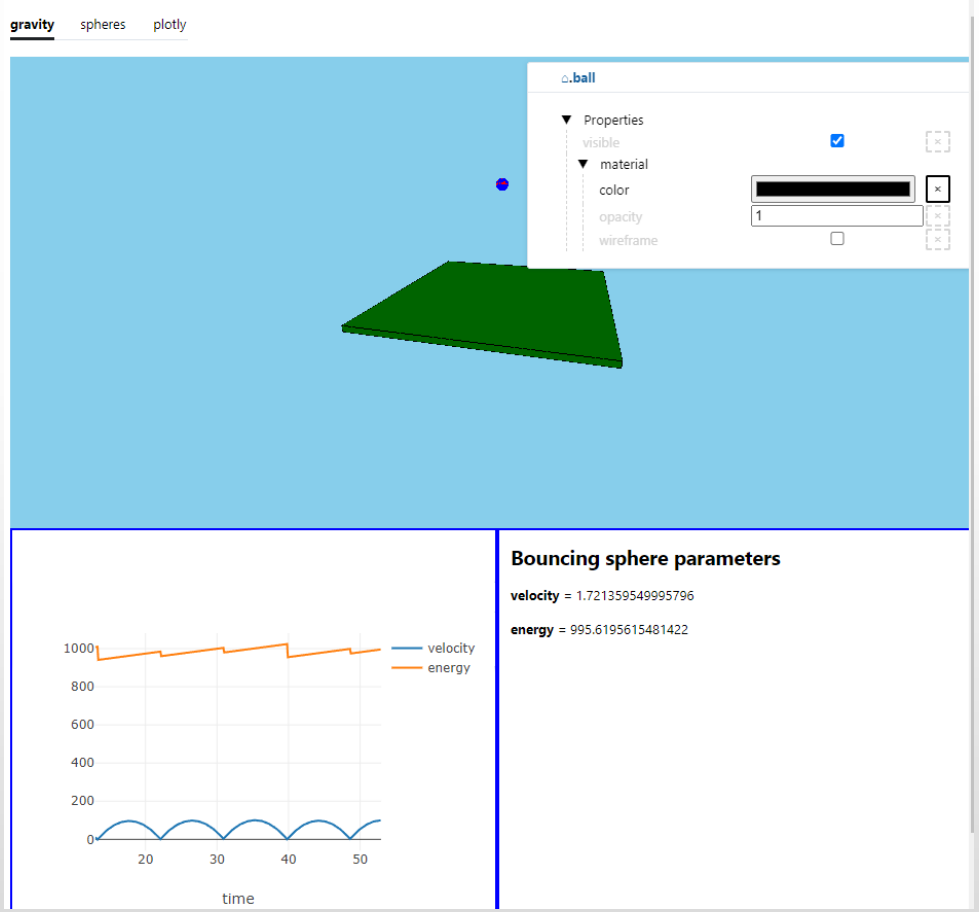
VisionForge modules



- Core functionality and properties modules.
- Object model for 3D.
- Three-JS 3D renderer.
- Plotly integration (via Plotly-kt) for 2D and 3D plots).
- Tables rendering.
- Markup rendering.

Not only 3D - dashboards

https://npm.mipt.ru/demos/vf-dynamic/



https://github.com/SciProgCentre/visionforge

- 3D rendering with three-js
- Dynamic plot with Plotly-kt
- Dynamic markdown rendering.

VisionForge-solid object model

- Arguments (like geometric parameters) passed in a constructor and can't be changed.
- Properties could be changed and are observed.
- Rotations are passed as Euler angles (like Root), rotation matrices or quaternions.
- Solid groups are true trees with name-based navigation.

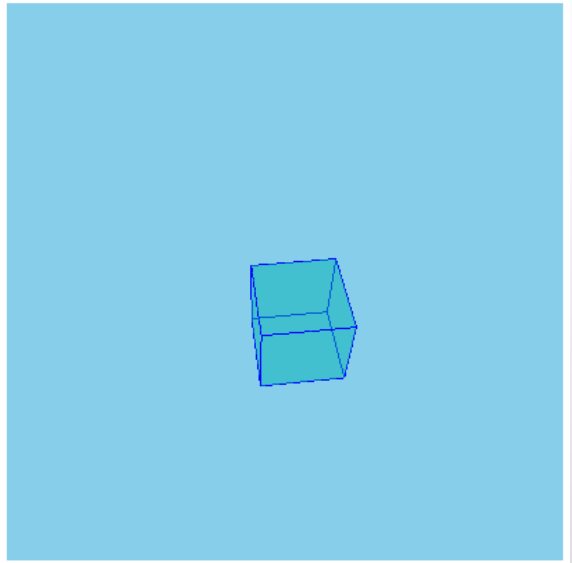
https://github.com/SciProgCentre/visionforge/blob/dev/docs/tutorials/tutorial-solids.md

Rendering a box

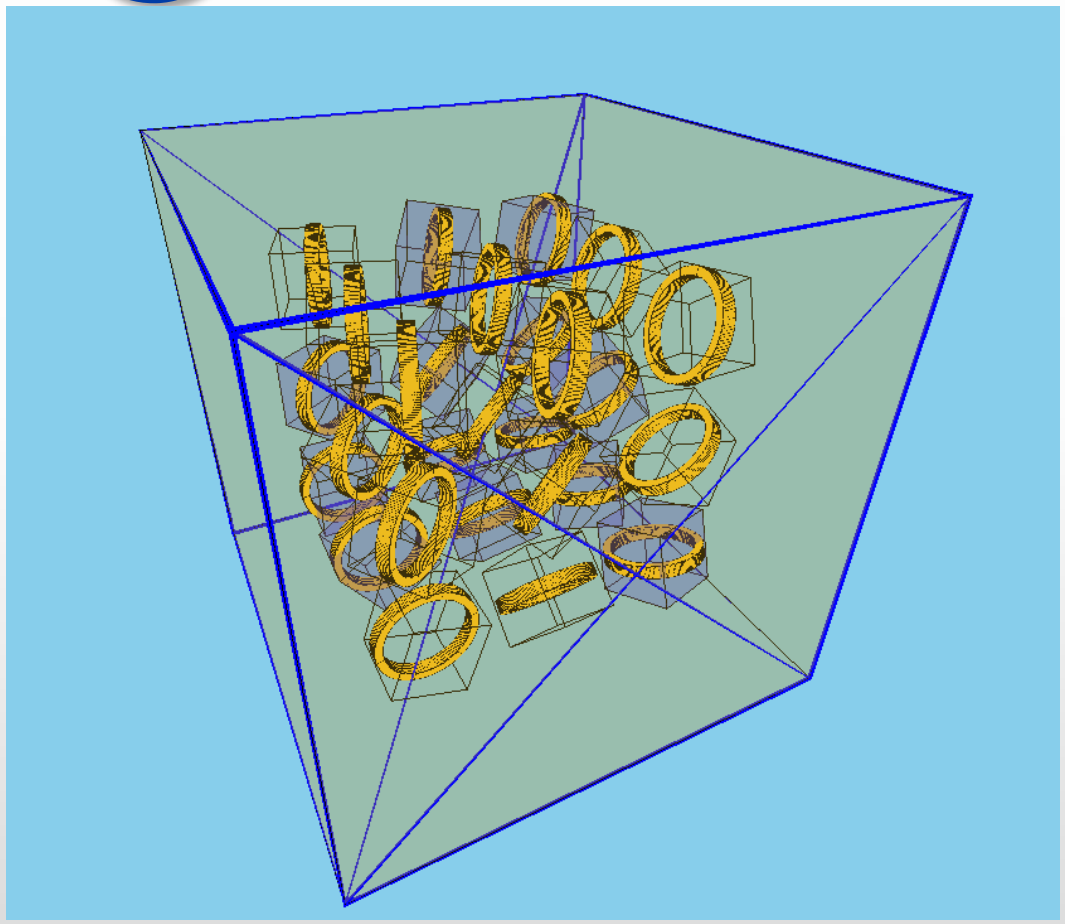
```
/**
 * A separate class is created to optimize native rendering
 */
 Alexander Nozik
@Serializable
@SerialName("solid.box")
public class Box(
    public val xSize: Float,
    public val ySize: Float,
    public val zSize: Float,
) : SolidBase<Box>(), Hexagon {

    Alexander Nozik
    private inline val dx get() = xSize / 2
    Alexander Nozik
    private inline val dy get() = ySize / 2
    Alexander Nozik
    private inline val dz get() = zSize / 2

    Alexander Nozik
    override val node1: Point3D get() = Point3D(-dx, -dy, -dz)
    Alexander Nozik
    override val node2: Point3D get() = Point3D(dx, -dy, -dz)
    Alexander Nozik
```

The `big box` will have properties with custom values.

```
box(40, 40, 40, name = "big box"){
   x = 20
   y = 10
   z = 60
   opacity = 0.5 //50% opacity
   color(0u, 179u, 179u) //color in rgb
   rotation = Point3D(60, 80, 0)
}
```

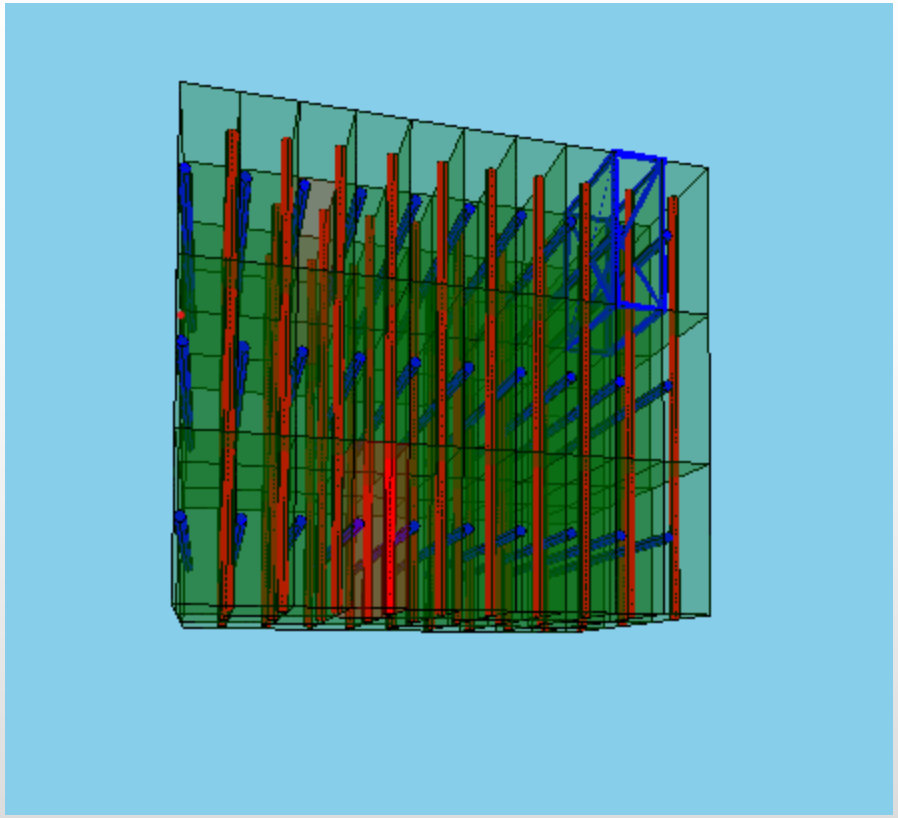


Convenient GDML geometry generator



```
public fun cubes(): Gdml = Gdml{
  val center = define.position(name = "center")
  structure {
    val air = materials.isotope("G4_AIR")
    val tubeMaterial = materials.element("tubeium")
    val boxMaterial = materials.element("boxium")

    val segment = solids.tube(20, 5.0, "segment") {
      rmin = 17
      deltaphi = 60
      aunit = AUnit.DEG
    }
    val worldBox = solids.box(200, 200, 200, "LargeBox")
    val smallBox = solids.box(30, 30, 30, "smallBox")
    val segmentVolume = volume(tubeMaterial, segment, "segment")
    val composite = volume(boxMaterial, smallBox, "composite") {
      for (i in 0 until 6) {
        physVolume(segmentVolume, "segment-$i") {
          positionref = center
          rotation {
            z = 60 * i
            unit = AUnit.DEG
          }
        }
      }
    }

    world = ...
  }
}
```

Patch-based updates



21:02:48.836 [DefaultDispatcher-worker-5] DEBUG ktor.application - Sending update for vision[1198130203]:

```
{
   "children": {
      "layer[7].segment[3,1]": {
         "properties": {
            "material": {
               "color": "red"
            }
         }
      }
   }
}
```

Full dot-separated name of a change target

Changed property value

CERN ROOT integration

Possible approaches:

- Read ROOT format and convert it to Vision Object Model.
- Create a ROOT plugin that converts TGeoManager to VOM on the ROOT side.
- Convert TGeoManager to JSON via TBufferJSON.

Limited success

Failed

Success

Future work

- Smart layering algorithm to render large geometries (like BM@N).
- Integration with BM@N software layout.
- Other rendering engines support (for example Scenery - https://github.com/scenerygraphics/scenery).