LISE ARIS 3D-Envelope: Panel Development

Alexandra Tarasova

New Isotope Research Group

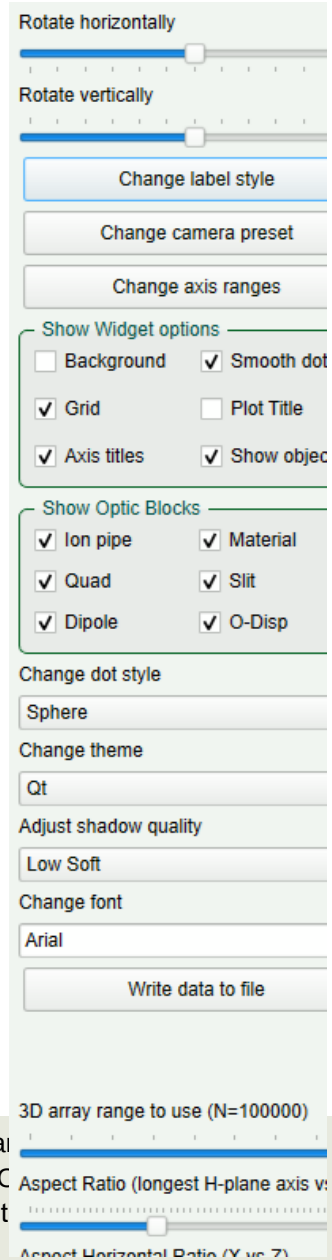
Facility for Rare Isotope Beams, Michigan State University, East Lansing, MI 48824 USA

05/02/25

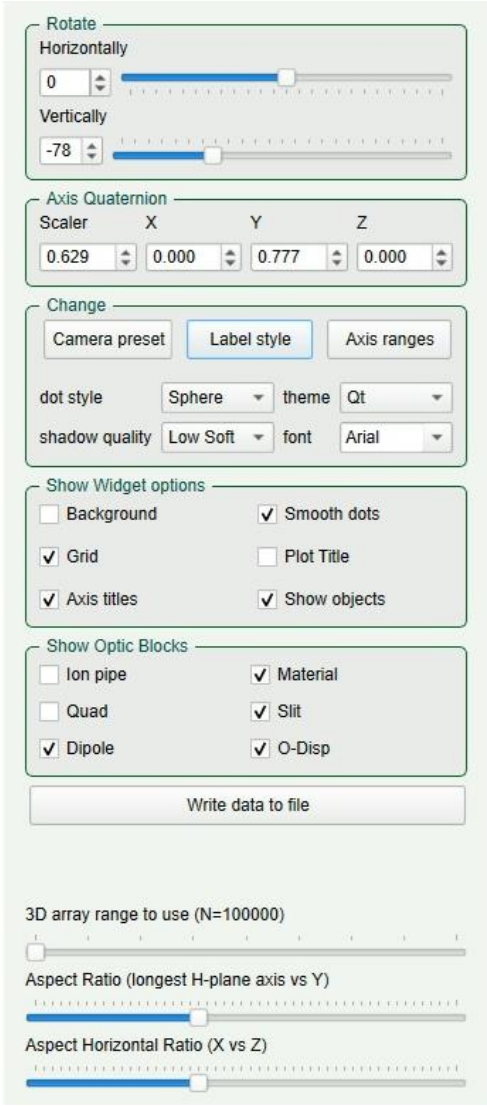
This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics and used resources of the Facility for Rare Isotope Beams (FRIB) Operations, which is a DOE Office of Science User Facility under Award Number DE-SC0023633, and by the US National Science Foundation under Grants No. PHY-20-12040 and 23-10078 “Windows on the Universe: Open Quantum Systems in Atomic Nuclei at FRIB”.

Panel Optimization

Previous design

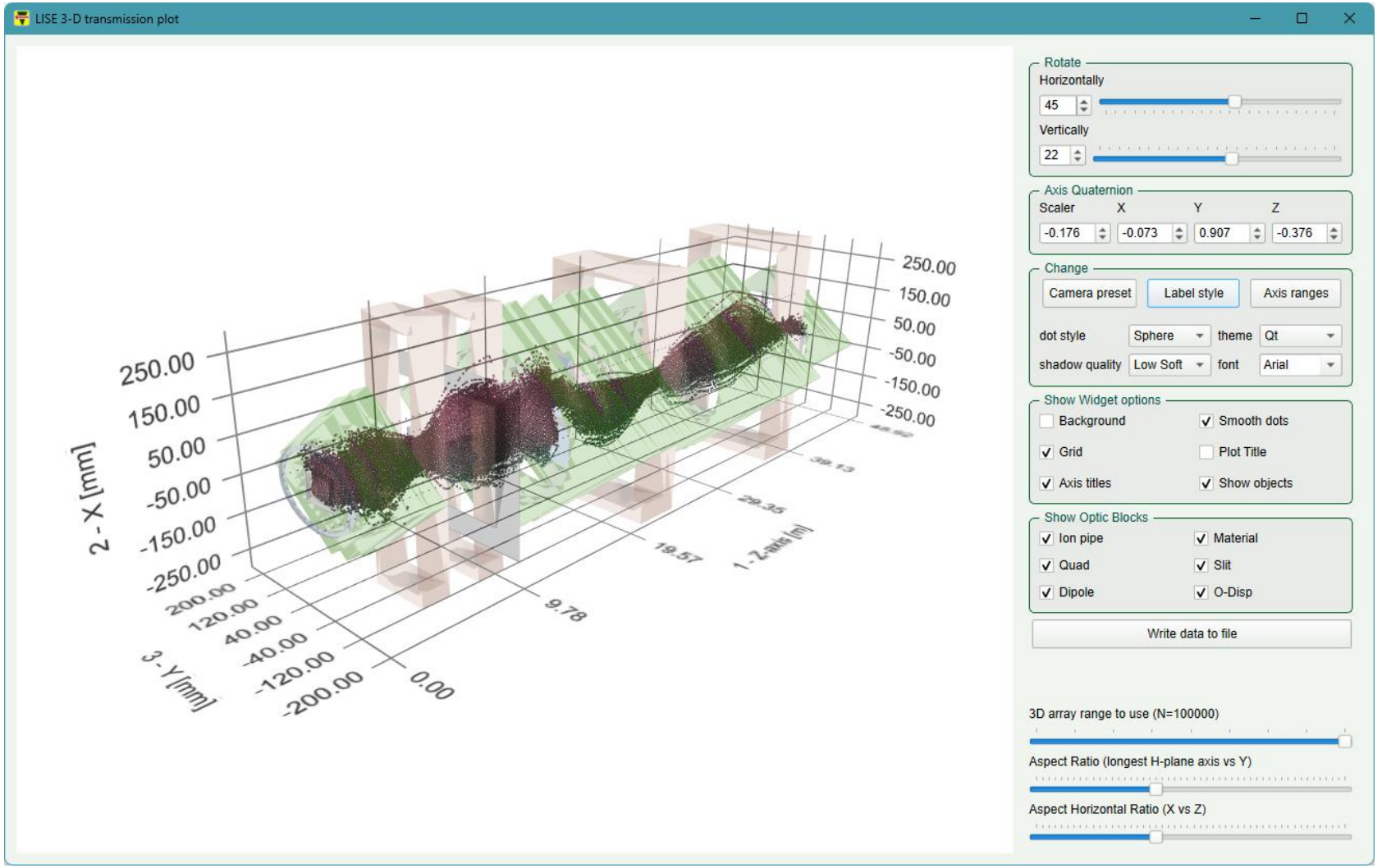


Optimized design with additional features

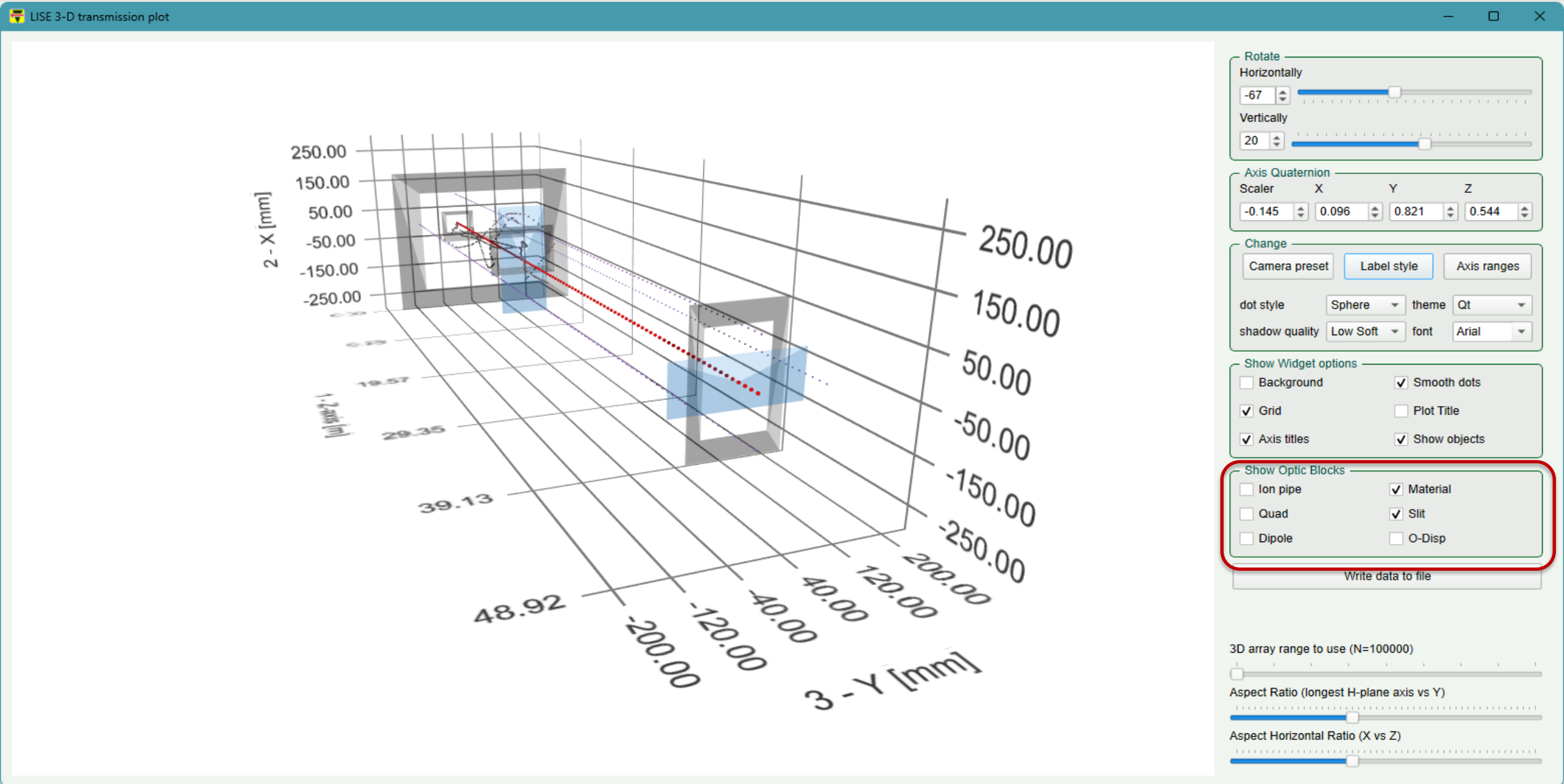


New Features

- Panning of the graph using the left mouse button
- Value input to adjust the angle of the graph, as well as arrows to increment.
- Axis Quaternion input boxes implemented to specify 3D rotation using quaternions.
- Angle of the graph updates the panel values and sliders in real time.
- Synchronization of manual rotation controls and quaternion-based camera orientation.

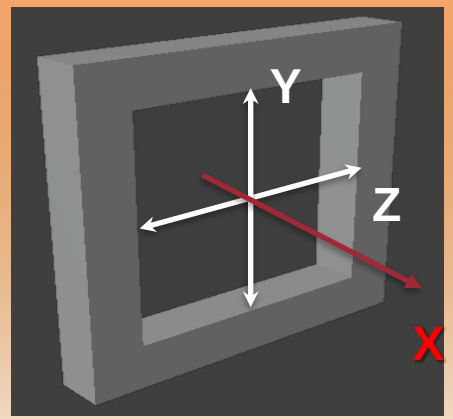


Plotting selected optic blocks



3D Separator Blocks Update (1)

Slit

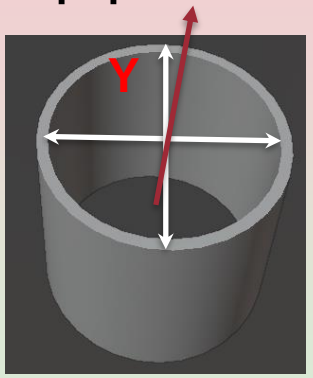


NEW
Length (thick) X = 0.02 m
Inner size Y & Z = 0.10 m
Outer size Y & Z = 0.14 m

Old

Dimensions:	
X	0.02 m
Y	0.14 m
Z	0.14 m

Ion pipe X

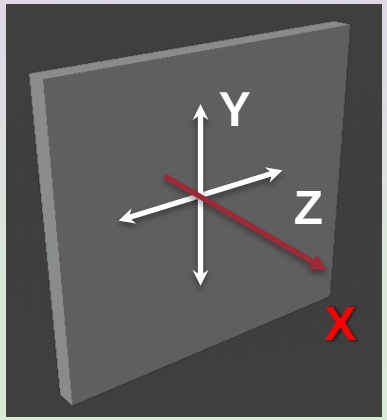


NEW
X – length : 0.2 m,
Y & Z Inner diameter 0.1 m
Y & Z Outer diameter 0.11 m

Old

Dimensions:	
X	0.11 m
Y	0.11 m
Z	0.1 m

Material



NEW
X length : 0.01 m,
Y size: 0.1 m
Z size : 0.1 m

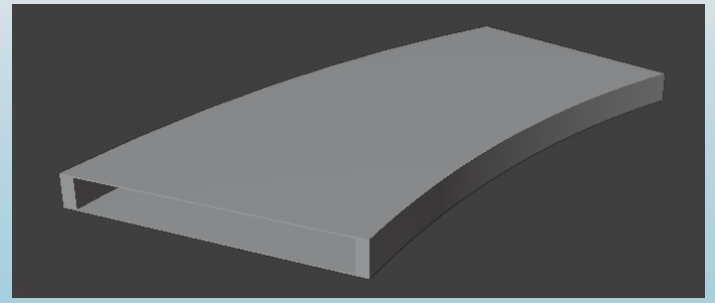
Old

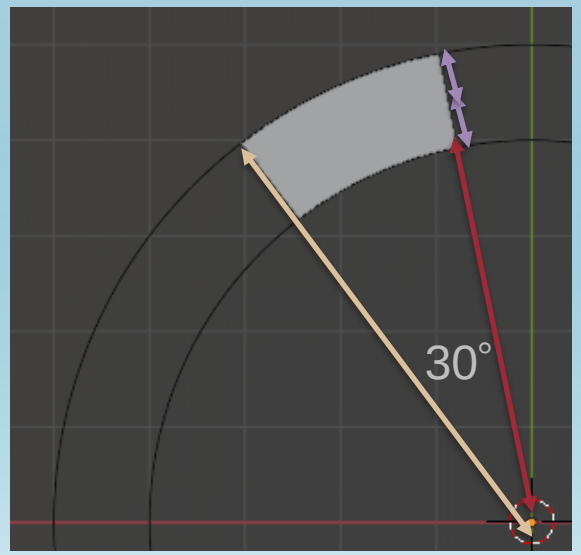
Dimensions:	
X	0.075 m
Y	0.075 m
Z	0.005 m

X is always the central axis for ALL blocks

3D Separator Blocks Update (2)

Dipole

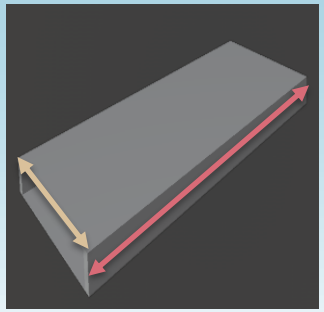


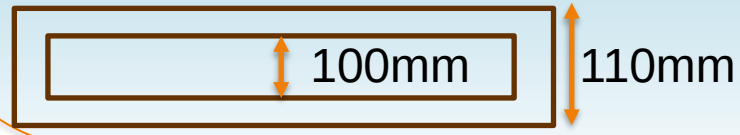


500 mm

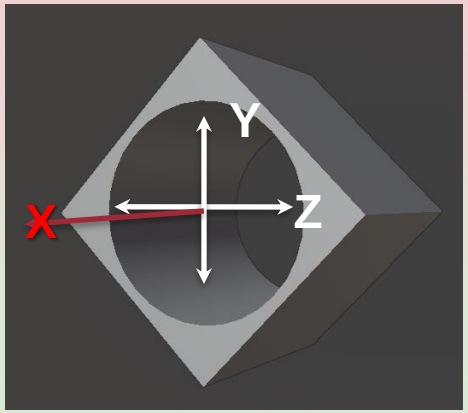
Rmax 5000mm

Rmin 4000mm





Quadrupole

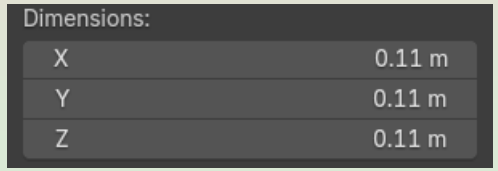


NEW

X length : 0.1 m,

Y-Z inner radius : 0.05 m

Old Inner radius 50mm



X is always the central axis for ALL blocks

Qt Quaternion & Camera

Qt Quaternion Overview

Definition

A quaternnion is a 4D construct for representing 3D rotations:

$$q = w + xi + yj, zk$$

- Quaternnion (float scalar, float *x*, float y, float, z)
- *w* serar = °: real part (w)
- *x, y, ty.* amatmiarry/vector *t*: par)

Why Use QUaternrion?

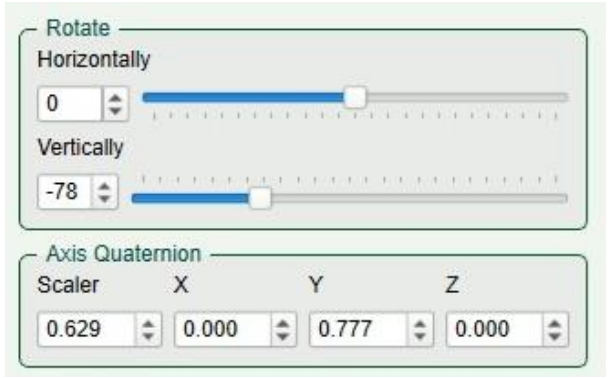
- Quaternnion();
- Quaternnion*x, w, x, y, z*);
- Quaternnion::fro*mAxisandAngle* (QVector3D axis, float angleDeg):
  Quaternnion::from*RotationMatrix* (QMatric3x3 matrix)

Essential Methods

Mcthod	Purpose
scalar()	Gcalar)
x(), y, z)	Get imaginarry part
normalized()	Rotate vector v using qoia
conjugated()	Reverse rotation
inverted()	Inverse of the rotation
slerp (q1, q2 .t)	Smooth interpolation bVv

Example

```
QQuaternnion q = Quaternnion::fromAxisAndAngleQVvectorD(0, 1, 0), 90);
QVector3D v = q.rotatedVectorQVvectorD(1, 0, 0));
/→ (0, 0, −1)
```



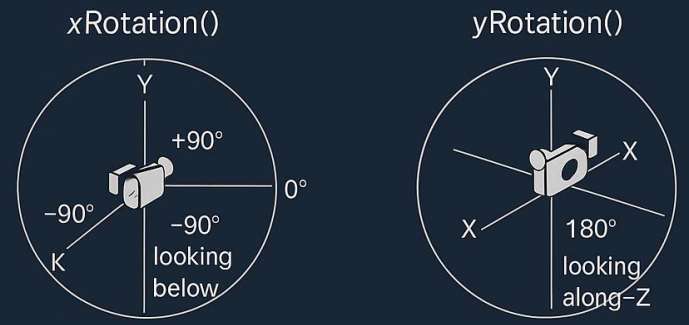
How to synchronize them in code?

Qt Camera: X & Y Rotations

Q3DCamera handles 3D viewing using two anges: Ax, YRotation()

Property	Meaning	Axis of Rotation
*x*Rotation()	Vertical tilt (elevation)	X
yRotation()	Horizontal pan (azimuth)	Y

Visual Meaning:



Example:

```
Q3DCamera *cam = myJDView->scene()->activeCamera()
```

Qt Quaternion & Camera: synchronization (1)

Quaternion

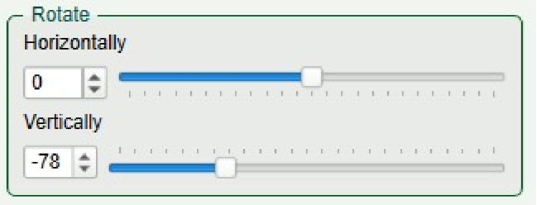

```
QObject::connect(spinS,  &QDoubleSpinBox::valueChanged, this, &TGraph3D::CmAngleRotations);
QObject::connect(spinX,  &QDoubleSpinBox::valueChanged, this, &TGraph3D::CmAngleRotations);
QObject::connect(spinY,  &QDoubleSpinBox::valueChanged, this, &TGraph3D::CmAngleRotations);
QObject::connect(spinZ,  &QDoubleSpinBox::valueChanged, this, &TGraph3D::CmAngleRotations);
```

Implementation of special variable :
flagPermitAngleChanged

```
void TGraph3D::CmAngleRotations(double )
{
  if(!flagPermitAngleChanged) return;

  modifier.get()->rotationAngles.setScalar(spinS->value());
  modifier.get()->rotationAngles.setX(spinX->value());
  modifier.get()->rotationAngles.setY(spinY->value());
  modifier.get()->rotationAngles.setZ(spinZ->value());

// Convert to forward (view direction) vector
QVector3D forward = -(modifier.get()->rotationAngles).rotatedVector(QVector3D(0, 0, 1));

// Normalize to be safe
forward.normalize();

// Compute spherical angles
float xRad = qAsin(forward.y()); // elevation
float yRad = qAtan2(forward.x(), forward.z()); // azimuth

float xRotationDeg = qRadiansToDegrees(xRad); // pitch
float yRotationDeg = qRadiansToDegrees(yRad); // yaw

if (yRotationDeg < 0)
    yRotationDeg += 360.0f;


  flagPermitAngleChanged = false;

Q3DCamera *cam = graph->scene()->activeCamera();
cam->setXRotation(xRotationDeg);
cam->setYRotation(yRotationDeg);

  flagPermitAngleChanged = true;
}
```

Qt Quaternion & Camera: synchronization (2)

Camera



SpinBox & Slider

```
QObject::connect(rotationSliderX, &QSlider::valueChanged, modifier.get(), &ScatterData::rotateX);
QObject::connect(rotationSpinX,  qOverload<int>(&QSpinBox::valueChanged), rotationSliderX, &QSlider::setValue);
QObject::connect(rotationSliderX, &QSlider::valueChanged, rotationSpinX, &QSpinBox::setValue);

QObject::connect(rotationSliderY, &QSlider::valueChanged, modifier.get(), &ScatterData::rotateY);
QObject::connect(rotationSpinY,  qOverload<int>(&QSpinBox::valueChanged), rotationSliderY, &QSlider::setValue);
QObject::connect(rotationSliderY, &QSlider::valueChanged, rotationSpinY, &QSpinBox::setValue);
```

Rotated by clicking mouse right button of image:

```
QObject::connect(modifier.get(), &ScatterData::rotateXchanged, rotationSliderX, &QSlider::setValue);
QObject::connect(modifier.get(), &ScatterData::rotateYchanged, rotationSliderY, &QSlider::setValue);
```

Implementation of special variable : **flagRotate**

```
void ScatterData::rotateX(int rotation)
{
  Q3DCamera *camera = m_graph->scene()->activeCamera();

  m_xRotation = rotation;
  if(flagRotate <=0 )
    camera->setCameraPosition(m_xRotation, m_yRotation);
  else flagRotate--;

  calculateCamRotation();
}
//WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
void ScatterData::rotateY(int rotation)
{
  m_yRotation = rotation;
   if(flagRotate==0)
       m_graph->scene()->activeCamera()->setCameraPosition(m_xRotation, m_yRotation);
   else flagRotate--;

   calculateCamRotation();
}
```

```
void ScatterData::calculateCamRotation()
{
  // Convert to radians
  float xRad = qDegreesToRadians(m_xRotation);
  float yRad = qDegreesToRadians(m_yRotation);

  // Forward vector (from camera to target)
  QVector3D forward(
      qCos(xRad) * qSin(yRad),
      qSin(xRad),
      qCos(xRad) * qCos(yRad)
  );

  // Right vector (camera's local X axis)
  QVector3D up(0.0f, 1.0f, 0.0f);  // World up
  QVector3D right = QVector3D::crossProduct(forward, up).normalized();

  // Recomputed up (camera's local Y axis)
  QVector3D cameraUp = QVector3D::crossProduct(right, forward).normalized();

  // Optionally: flip forward for OpenGL-like direction
  QVector3D cameraZ = -forward.normalized();

  QMatrix3x3 rotMatrix;
  setColumn(rotMatrix, 0, right);      // X axis
  setColumn(rotMatrix, 1, cameraUp);   // Y axis
  setColumn(rotMatrix, 2, cameraZ);    // Z axis

  rotationAngles = QQuaternion::fromRotationMatrix(rotMatrix);
  emit dataAngleChanged(rotationAngles);
}
```

Final Animation Page