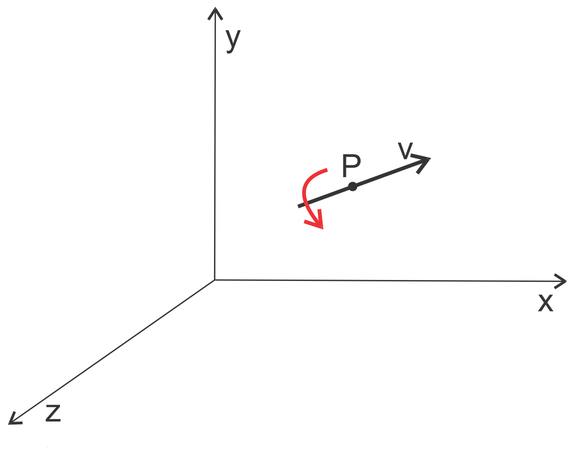
Слободна ротација. Торус. Коса пирамида

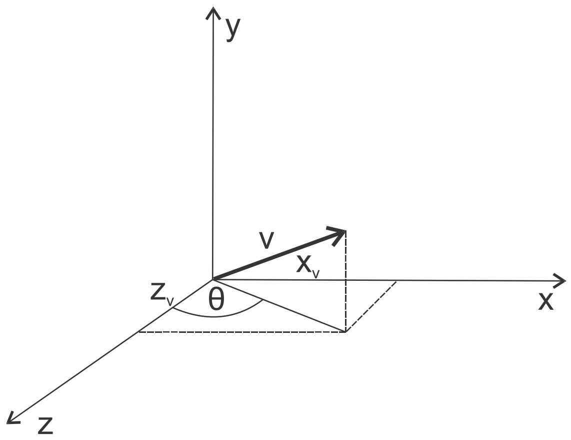
Ротација око произвољне осе која пролази кроз произвољну тачку

Потребно је креирати матрицу ротације за угао α око произвољне осе која пролази кроз тачку $P(x_P, y_P, z_P)$ и има правац вектора $\vec{v}(x_v, y_v, z_v)$ (Слика 1).



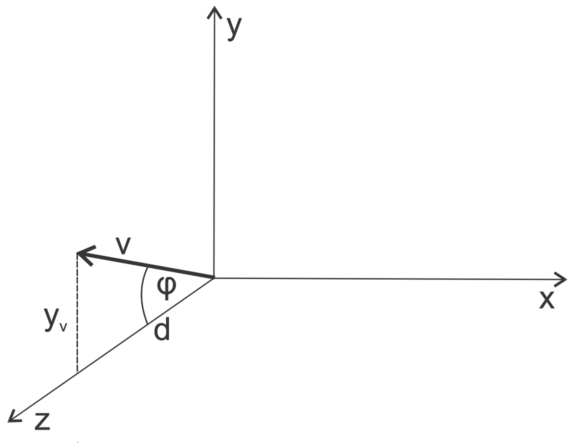
Слика 1. Ротација око произвољне осе

Да би се извршила оваква ротација потребно је довести модел који се ротира у такав положај да се може извршити ротација око неке од координатних оса, а потом извршити ротацију за угао α око те осе. Прво се модел транслира у координатни координатни почетак, односно за вектор $-\vec{P}$ (транслација $T(-x_P, -y_P, -z_P)$). На слици 2 је сада тачка $P(x_P, y_P, z_P)$ транслиран у координатни почетак.



Слика 2. Вектор $\vec{v}$ транслиран у координатни почетак

Сада се модел ротира за угао $-\theta$ око y осе тако да се након ротације нађе у равни yOz. Угао θ једнак је $\theta = arctg\left(\frac{x_V}{z_v}\right)$ уколико се вектор не налази у равни xOy, а уколико је $z_v = 0$ тада је $\theta = \frac{\pi}{2}$ за $x_v > 0$ или $\theta = -\frac{\pi}{2}$ за $x_v < 0$. Ако је и $x_v = 0$ тада је $\theta = 0$. Та матрица ротације се може обележити са $R_{y\_axis,-\theta}$.



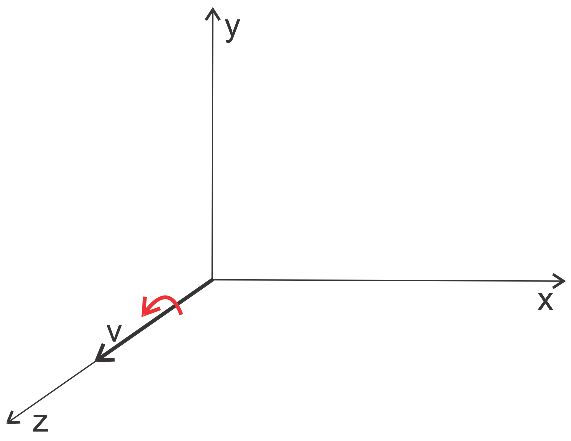
Слика 3. Вектор $\vec{v}$ у равни yOz

Вектор $\vec{v}$ се након ове ротације налази у равни yOz као на слици 3. Даље, да би се вектор $\vec{v}$ поклопио са z осом потребно је још ротирати га око x осе за угао φ где је

$$\varphi = \begin{cases} arctg\left(\frac{y_V}{d}\right), & d \neq 0 \\ \frac{\pi}{2}, & d = 0 \wedge y > 0 \\ -\frac{\pi}{2}, & d = 0 \wedge y < 0 \end{cases}$$

$$d = \sqrt{x_v^2 + z_v^2}\ .$$

Матрица тако дефинисане ротације може се означити са $R_{x\_axis,\varphi}$. Сада се вектор $\vec{v}$ поклапа са z осом као на слици 4, па се може извршити ротација модела за угао α око z осе. Таква ротација се може означити са $R_{z\_axis,\alpha}$.



Слика 4. Вектор $\vec{v}$ поклопљен са z осом

Када се изврши ротација за угао α, потребно је применити такве трансформације да се модел врати у почетни положај. Потребно је прво извршити ротације $R_{x\_axis,-\varphi}$ и $R_{y\_axis,\theta}$, а потом и транслацију $T\left(x_P, y_P, z_P\right)$. Тако се коначна матрица ротације добија множењем наведених седам матрица трансформације и то

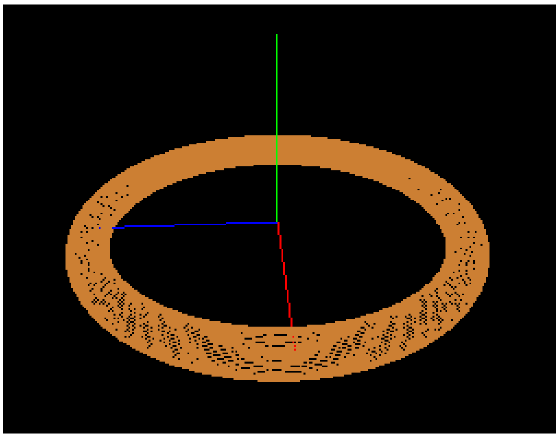
$$R_{\vec{v},\alpha} = T_{x_P,y_P,z_P} \cdot R_{y\_axis,\theta} \cdot R_{x\_axis,-\varphi} \cdot R_{z\_axis,\alpha} \cdot R_{x\_axis,\varphi} \cdot R_{y\_axis,-\theta} \cdot T_{-x_P,-y_P,-z_P} .$$

Функција за креирање овако дефинисане ротације може се видети у додатку на крају документа.

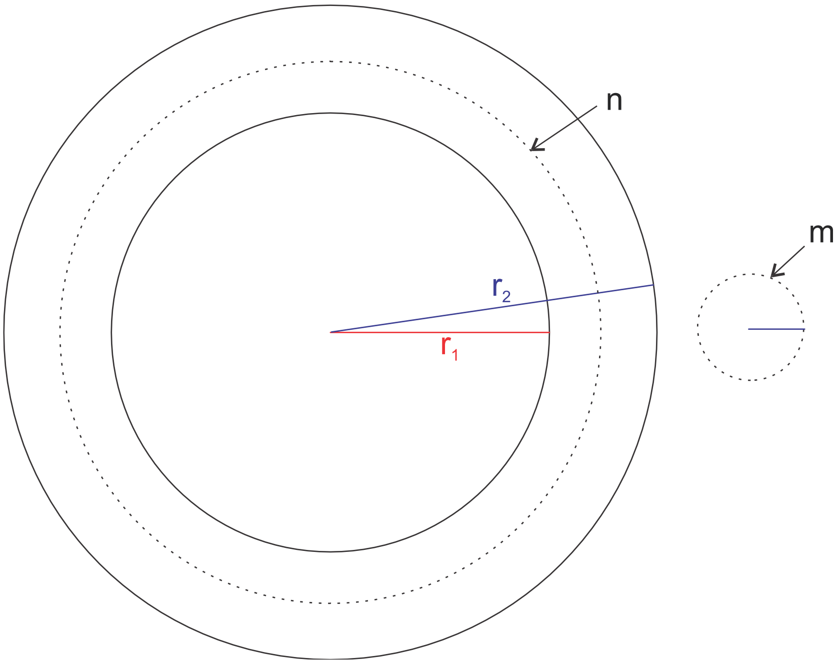
Опис проблема

1. Направити функцију за цртање торуса са следећим карактеристикама:

- Центар торуса треба да се нађе у координатном почетку
- Центри свих кружница којима се оивичава торус треба да леже у равни xOz.
- Задати су и следећи параметри: r_1 - унутрашњи полупречник торуса, r_2 - спољашњи полупречник торуса, n - број кружница којима ће бити оивичен торус и m - број тачака по кружницама којима је торус оивичен (Слика 6).



Слика 5. Торус



Слика 6. Параметри торуса

2. Нацртати четворострану праву пирамиду задату на следећи начин:

- Дужина странице у основи је a.
- Висина пирамиде је h.
- Нормала равни у којој лежи основа пирамиде је n.
- Пресек дијагонала основе треба да лежи у координатном почетку.
- Пројекције две странице основе пирамиде на xOz раван треба да буду паралелне са x осом, а пројекције друге две странице паралелне са z осом.

Решење

Помоћне функције за трансформацију једнодимензионалног и дводимензионалног вектора чворова за задату матрицу трансформације:

```
void TransformV1(Matrix4x4 &M, vector<Vector3D> &v)
{
    for(unsigned int i = 0; i < v.size(); i++)
        v[i] = M.Transform(v[i]);
}

void TransformV2(Matrix4x4 &M, vector< vector<Vector3D> > &v)
{
    for(unsigned int i = 0; i < v.size(); i++)
        TransformV1(M,v[i]);
}
```

1. Прво је креирана кружница у равни xOy са центром у тачки $C\left(0.5\cdot\left(r\_inside+r\_outside\right),0,0\right)$ и полупречником $0.5\cdot\left(r\_outside-r\_inside\right)$. Нека је $\psi=\dfrac{2\cdot\pi}{m}$ где је *m* број кружница којима ће торус бити оивичен. Сада се формира низ од *n* кружница копирањем и ротирањем добијене кружнице редом за углове 0, ψ, 2ψ, ..., $\left(n-1\right)\psi$. Тиме је конструисан костур торуса састављен од *n* кружница. Спајањем тачака суседних кружница слично омотачу ваљка добија се торус.

```
void CreateTorus(vector< vector<Vector3D> > &torus, double r_inside,
                 double r_outside, int n, int m)
{
    double r1 = 0.5*(r_inside  + r_outside);
    double r2 = 0.5*(r_outside - r_inside);

    torus.resize(n);
    for(int i = 0; i < n; i++)
        torus[i].resize(m);

    Matrix4x4 MRotate1, MRotate2, MT1, MT2, M;
    MRotate1.loadRotateY(2.0*M_PI/n);

    MT1.loadTranslate(-r1,0.0,0.0);
    MRotate2.loadRotateZ(2.0*M_PI/m);
    MT2.loadTranslate(r1,0.0,0.0);
    M = MT2*MRotate2*MT1;

    torus[0][0] = Vector3D(r1+r2, 0.0, -r2);
    for(int i = 1; i < m; i++)
        torus[0][i] = M.Transform(torus[0][i-1]);

    for(int i = 1; i < n; i++)
    {
        torus[i] = torus[i-1];
```

```
        TransformV1(MRotate1, torus[i]);
    }
}

void DrawCylinderWrapper(vector< vector<Vector3D> > &cylinder)
{
    vector<Vector3D> poly;
    poly.resize(4);

    int n = cylinder[0].size();
    for(int i = 0; i < n-1; i++)
    {
        poly[0] = cylinder[0][i];
        poly[1] = cylinder[0][i+1];
        poly[2] = cylinder[1][i+1];
        poly[3] = cylinder[1][i];

        DrawPolygon(poly);
    }

    poly[0] = cylinder[0][n-1];
    poly[1] = cylinder[0][0];
    poly[2] = cylinder[1][0];
    poly[3] = cylinder[1][n-1];

    DrawPolygon(poly);
}

void DrawTorus(vector< vector<Vector3D> > &torus)
{
    vector< vector<Vector3D> > t;
    t.resize(2);
    for(unsigned int i = 0; i < torus.size()-1; i++)
    {
        t[0] = torus[i];
        t[1] = torus[i+1];
        DrawCylinderWrapper(t);
    }

    t[0] = torus[torus.size()-1];
    t[1] = torus[0];
    DrawCylinderWrapper(t);
}
```

2. Објекат је најлакше креирати око координатног почетка, а тек потом поставити објекат на одговарајуће место на задати начин. Зато се чворови пирамиде прво креирају тако да се пресек дијагонала основе нађе у координатном почетку, да странице основе буду паралелне осама x и z, и да се врх пирамиде нађе на позитивном делу y осе и то на растојању h од координатног почетка.

Нека је α угао који нормала n заклапа са y осом и нека је вектор $\vec{l} = \vec{n} \times \vec{y}$. Такође нека је права k одређена координатним почетком и вектором $\vec{l}$. Тада, да би се основа пирамиде нашла у равни одређеној нормалом n и координатним почетком, потребно је ротирати је за угао α око праве k.

Коначно, потребно је да по две странице основе пирамиде буду паралелне осама x и z. Нека је сада страница AB била паралелна z оси пре претходно извршене ротације. Нека је φ угао између пројекције странице AB на раван xOz и z осе. Сада се пирамида ротира за угао φ око осе која пролази кроз координатни почетак а има правац вектора n.

Функција `AngleBetween` рачуна угао између два вектора, и њен код се може видети у додатку на крају ове скрипте.

```
void drawPyramidTransform(Vector3D &n, double a, double h)
{
    Vector3D H;
    H = Vector3D::AxisY * h;

    Vector3D A(0.5*a,0.0,0.5*a);
    Vector3D B(0.5*a,0.0,-0.5*a);
    Vector3D C(-0.5*a,0.0,-0.5*a);
    Vector3D D(-0.5*a,0.0,0.5*a);

    Vector3D yAxis = Vector3D::AxisY;
    double alpha = AngleBetween(n, yAxis);
    Vector3D axis = n.Cross(Vector3D::AxisY);
    Matrix4x4 MT;
    Vector3D O(0,0,0);
    LoadRotationMatrix(O,axis,-alpha,MT);

    H = MT.Transform(H);
    A = MT.Transform(A);
    B = MT.Transform(B);
    C = MT.Transform(C);
    D = MT.Transform(D);

    Vector3D ab;
    ab = B - A;
    ab.m_y = 0.0;
    Vector3D zAxis = Vector3D::AxisZ;
    double fi = AngleBetween(ab,zAxis);
    LoadRotationMatrix(O,n,fi,MT);

    A = MT.Transform(A);
    B = MT.Transform(B);
    C = MT.Transform(C);
    D = MT.Transform(D);

    glBegin(GL_QUADS);
        glColor3f(1.0f, 1.0f, 0.0f);      // Yellow
        glVertex3f( A.m_x, A.m_y, A.m_z);
```

```
        glVertex3f( B.m_x, B.m_y, B.m_z);
        glVertex3f( C.m_x, C.m_y, C.m_z);
        glVertex3f( D.m_x, D.m_y, D.m_z);
    glEnd();

    glBegin(GL_TRIANGLE_FAN);
        glColor3f(1.0f, 1.0f, 0.0f);     // Yellow
        glVertex3f( H.m_x, H.m_y, H.m_z);
        glVertex3f( A.m_x, A.m_y, A.m_z);
        glVertex3f( B.m_x, B.m_y, B.m_z);
        glVertex3f( C.m_x, C.m_y, C.m_z);
        glVertex3f( D.m_x, D.m_y, D.m_z);
        glVertex3f( A.m_x, A.m_y, A.m_z);
    glEnd();
}
```

Додатак

```
void LoadRotationMatrix(Vector3D &p, Vector3D &v, double alpha)
{
    Matrix4x4 tranP, tranPMinus, rotateFi, rotateFiMinus, rotateAlfa,
rotateTeta, rotateTetaMinus;

      double teta, fi, d;
      d = sqrt(v.X()*v.X() + v.Z()*v.Z());

      if(v.X() == 0) teta = 0;
      else
      {
            if (v.Z() != 0) teta = atan(v.X() / v.Z());
            else
            {
                  if (v.X() > 0) teta = M_PI / 2;
                  else teta = -M_PI / 2;
            }
      }

      if(v.Y() == 0) fi = 0;
      else
      {
            if (d != 0) fi = atan(v.Y() / d);
            else
            {
                  if (v.Y() > 0) fi = M_PI / 2;
                  else fi = -M_PI / 2;
            }
      }

      tranP.loadTranslate(p.X(), p.Y(), p.Z());
      tranPMinus.loadTranslate(-p.X(), -p.Y(), -p.Z());

      rotateFi.loadRotateX(fi);
      rotateFiMinus.loadRotateX(-fi);

      rotateTeta.loadRotateY(teta);
      rotateTetaMinus.loadRotateY(-teta);

      rotateAlfa.loadRotateZ(alpha);

      *this = tranP * rotateTeta * rotateFiMinus * rotateAlfa * rotateFi *
rotateTetaMinus * tranPMinus;
}
```

```
double AngleBetween(Vector3D &a, Vector3D &b)
{
    double kosFi = (a * b) / (a.Norm() * b.Norm());
    double sinFi = (a.Cross(b)).Norm() / (a.Norm() * b.Norm());

    double fi = atan2(sinFi,kosFi);

    return fi;
}
```