BÀI 3
MÔ PHỎNG CHUYỂN ĐỘNG CỦA ROBOT VỚI OPENGL

Tóm tắt

Trong bài này, các kỹ thuật sau sẽ được giới thiệu

- Đọc và hiển thị các tệp STL (được xuất ra từ AutoCAD, Solidworks…)
- Sử dụng ma trận đồng nhất để hiển thị vị trí các đối tượng hình học trong không gian 3 chiều
- Mô phỏng chuyển động của robot bằng OpenGL

Nội dung

1. Đọc và hiển thị các tệp STL	• Cấu trúc file ASCII STL • Lớp CSTL_File • Hiển thị đối tượng trong tệp STL
2. Sử dụng ma trận đồng nhất trong mô phỏng	• Mô hình hệ mặt trời-trái đất-mặt trăng • Ma trận chuyển vị đồng nhất • Sử dụng các phép dịch chuyển đơn lẻ
3. Mô phỏng chuyển động của robot bằng OpenGL	• Các bước thực hiện • Minh họa: Mô phỏng cơ cấu 4 khâu • Bài tập: Mô phỏng robot Scorbot

Cấu trúc file ASCII STL

solid AutoCAD
  facet normal 0.0000000e+000 0.0000000e+000 1.0000000e+000
    outer loop
      vertex 1.0000010e+000 1.0000010e+000 1.0000010e+000
      vertex 1.0000000e-006 1.0000010e+000 1.0000010e+000
      vertex 1.0000010e+000 1.0000000e-006 1.0000010e+000
    endloop
  endfacet
…
  facet normal 1.0000000e+000 0.0000000e+000 0.0000000e+000
    outer loop
      vertex 1.0000010e+000 1.0000000e-006 1.0000000e-006
      vertex 1.0000010e+000 1.0000010e+000 1.0000000e-006
      vertex 1.0000010e+000 1.0000010e+000 1.0000010e+000
    endloop
  endfacet
endsolid AutoCAD

Cấu trúc file STL

Dạng văn bản

- solid *name*
- *Danh sách tam giác*
  - facet normal n_i n_j n_k
  - outer loop
    - vertex $v1_x$ $v1_y$ $v1_z$
    - vertex $v2_x$ $v2_y$ $v2_z$
    - vertex $v3_x$ $v3_y$ $v3_z$
    - endloop
  - endfacet
- endsolid *name*

Dạng nhị phân

- UINT8[80] – Tiêu đề
- UINT32 – Số tam giác
- Danh sách tam giác
  - REAL32[3] – véc-tơ pháp
  - REAL32[3] – Đỉnh 1
  - REAL32[3] – Đỉnh 2
  - REAL32[3] – Đỉnh 3
  - UINT16 – Thuộc tính

Lớp CSTL_File

- Là một lớp đối tượng C++ dùng để thao tác các tệp STL
- Lấy từ project AMF tại http://amff.wikispaces.com/STL+to+AMF+converter
- Các thao tác chính
  - Đọc file STL (nhị phân & văn bản) vào bộ nhớ
  - Tính toán hình hộp bao
  - Vẽ đối tượng bằng các lệnh OpenGL

Lớp CSTL_File

```
class CSTL_File
{
public:
    CSTL_File(void);
    ~CSTL_File(void);
    bool Save(std::string filename, bool Binary = true) const;
    void ComputeBoundingBox (Vec3D& pmin, Vec3D& pmax);
    int Size() const;
    bool Load(std::string filename);
    bool LoadBinary(std::string filename);
    bool LoadAscii(std::string filename);
    void Draw(bool bModelhNormals, bool bShaded);
};
```

Lớp CSTL_File

- Đọc tệp STL vào bộ nhớ

  bool Load(std::string filename);

  bool LoadBinary(std::string filename);

  bool LoadAscii(std::string filename);

- Tìm hình hộp chứa trọn vật thể

  ComputeBoundingBox (Vec3D& pmin, Vec3D& pmax);

- Vẽ đối tượng bằng OpenGL

  void Draw(bool bModelhNormals, bool bShaded);

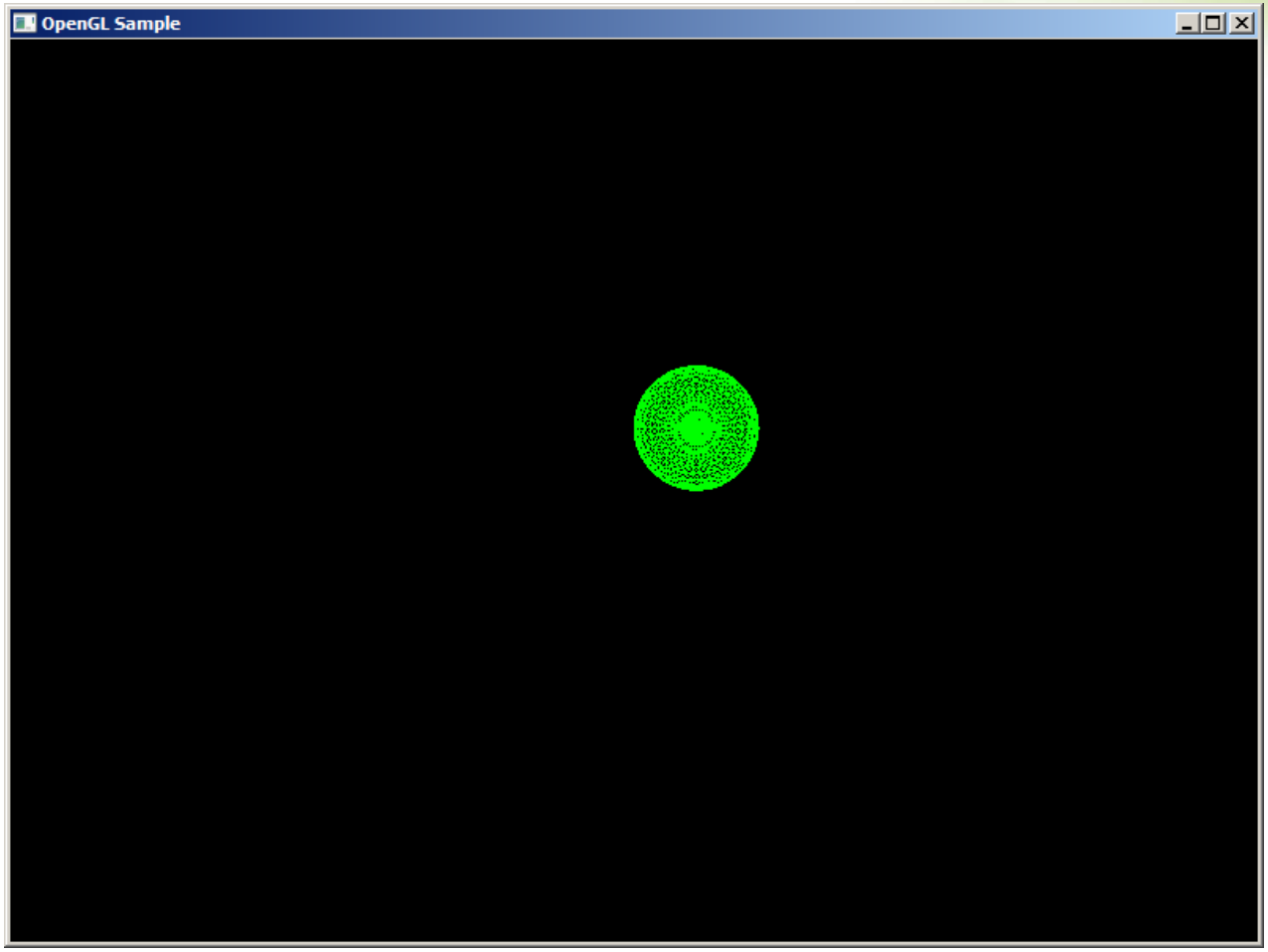
  bModelhNormals = true: vẽ véc-tơ pháp

  bShaded = true: vẽ mặt trơn thay vì khung dây

Sử dụng Lớp CSTL_File để hiển thị đối tượng trong tệp STL

1. Copy 3 file Vec3D.h, STL_File.h và STL_File.cpp vào thư mục project
2. Thêm 3 file Vec3D.h, STL_File.h và STL_File.cpp vào project
3. Thêm dòng #include "STL_File.h" ở đầu chương trình
4. Thêm dòng CSTL_File stl; để khai báo biến stl
5. Thêm dòng stl.Load("sphere.stl"); vào trong hàm InitGraphics() để đọc tệp sphere.stl
6. Thêm hàm stl.Draw(false, false); để vẽ đối tượng

Kết quả



Dùng lớp CSTL_File vẽ nhiều đối tượng

- Nếu có nhiều đối tượng trong các tệp STL khác nhau cần vẽ thì cần khai báo thêm các biến:
  - CSTL_File stl1, stl2, stl3;

  hoặc
  - CSTL_File stl[3];

Tọa độ âm trong file STL

- Theo mặc định, các phần mềm CAD không cho phép xuất ra các tọa độ âm trong file STL. Điều này là để tạo ra các file STL tương thích với các máy in khắc hình (stereolithography).
- Nếu phát hiện tọa độ âm, phần mềm sẽ tự động tịnh tiến đối tượng để đảm bảo tọa độ xuất ra không âm. Điều này làm cho việc định vị đối tượng trong chương trình OpenGL có thể bị sai lệch.
- Một giải pháp là trong phần mềm CAD, dịch các đối tượng sao cho tọa độ dương, ghi nhận véc-tơ dịch chuyển, rồi trong chương trình OpenGL sử dụng lớp CSTL_File, truyền thêm tham số véc-tơ này cho hàm Load()

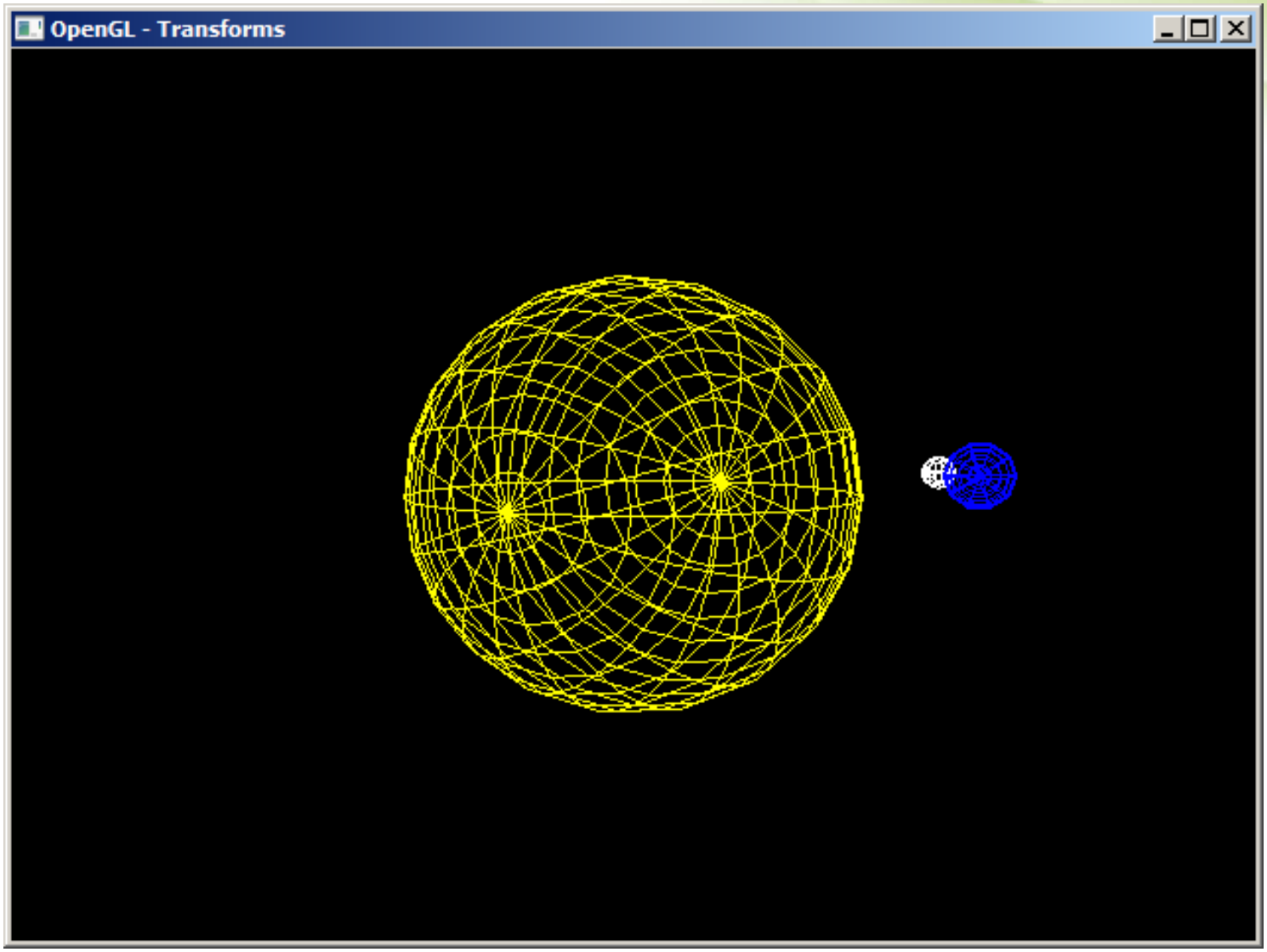
```
stl.Load("sphere.stl", Vec3D(100, 20, 50));
```

Câu hỏi?

Nội dung

1. Đọc và hiển thị các tệp STL	• Cấu trúc file ASCII STL • Lớp CSTL_File • Sử dụng lớp CSTL_File
2. Sử dụng ma trận đồng nhất trong mô phỏng	• Mô hình hệ mặt trời-trái đất-mặt trăng • Ma trận chuyển vị đồng nhất • Sử dụng các phép dịch chuyển đơn lẻ
3. Mô phỏng chuyển động của robot bằng OpenGL	• Các bước thực hiện • Minh họa: Mô phỏng cơ cấu 4 khâu • Bài tập: Mô phỏng robot Scorbot

Mô hình hệ mặt trời-trái đất-mặt trăng



Làm thế nào?

Sử dụng ma trận chuyển vị đồng nhất

Sử dụng ma trận chuyển vị đồng nhất 4x4

- Vẽ mặt trời

```
glMultMatrixf( mSunMatrix.m );

glColor4f( 1.0f, 1.0f, 0.0f, 1.0f );
renderWireSphere( 1.0f, 20, 20 );
```

- Vẽ trái đất

```
glMultMatrixf( mEarthMatrix.m );

glColor4f( 0.0f, 0.0f, 1.0f, 1.0f );
renderWireSphere( 1.0f, 10, 10 );
```

Vẽ mặt trời

```
matrix4x4f mSunMatrix;

mSunMatrix.rotate_y ( fSunSpin );

glPushMatrix();
{
   glMultMatrixf( mSunMatrix.m );
   glColor4f( 1.0f, 1.0f, 0.0f, 1.0f );
   renderWireSphere( 1.0f, 20, 20 );
}
glPopMatrix();
```

Vẽ trái đất

```
matrix4x4f mEarthTranslationToOrbit;
matrix4x4f mEarthSpinRotation;
matrix4x4f mEarthOrbitRotation;
matrix4x4f mEarthMatrix;

mEarthSpinRotation.rotate_y( fEarthSpin );

mEarthTranslationToOrbit.translate( vector3f(0.0f, 0.0f, -12.0f) );

mEarthOrbitRotation.rotate_y( fEarthOrbit );
```

```
mEarthMatrix =
  mEarthOrbitRotation *
  mEarthTranslationToOrbit *
  mEarthSpinRotation;

glPushMatrix();
{
   glMultMatrixf( mEarthMatrix.m );

   glColor4f( 0.0f, 0.0f, 1.0f, 1.0f );
   renderWireSphere( 1.0f, 10, 10 );
}
glPopMatrix();
```

Quy luật của hệ

```
static float fSunSpin    = 0.0f;  // Góc quay c
static float fEarthSpin  = 0.0f;  // Góc tự
static float fEarthOrbit = 0.0f; // Góc q
static float fMoonSpin   = 0.0f; // Góc tự
static float fMoonOrbit  = 0.0f; // Góc
```

ban đầ

ng q

```
fSunSpin -= g_fSpeedmodifier * (g_fElpasedTime * 10.0f);

fEarthSpin  -= g_fSpeedmodifier * (g_fElpasedTime * 100
fEarthOrbit -= g_fSpeedmodifier * (g_fElpasedTime * 20.0f

fMoonSpin  -= g_fSpeedmodifier * (g_fElpasedTime * 50.0
fMoonOrbit -= g_fSpeedmodifier * (g_fElpasedTime * 200.0f);
```

Hệ số tốc độ quay. Thay đổi bằng cách bấm F1/F2

Hệ số tốc độ quay của mặt

Hệ số tốc độ quay của trái

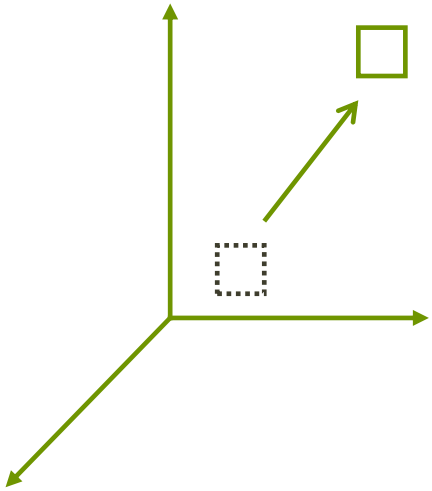
Hệ số tốc độ quay của mặt trăng quanh trái đất

Ma trận chuyển vị đồng nhất

- Là ma trận 4x4 để thực hiện phép biến đổi affine một cách đồng nhất: nhân ma trận $\mathbf{p}' = \mathbf{T}.\mathbf{p}$
- Các phép biến đổi affine chính: tịnh tiến, quay, phóng to-thu nhỏ, lấy đối xứng…
- Có dạng
- $$\mathbf{H} = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^T & \mathbf{1} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & t_1 \\ a_{21} & a_{22} & a_{23} & t_2 \\ a_{31} & a_{32} & a_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
- Trong đó $\mathbf{A}$ chính là ma trận cosin chỉ phương, $\mathbf{t}$ là véc-tơ tịnh tiến

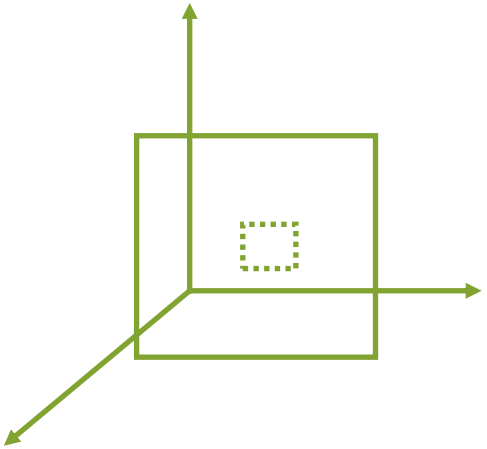
MỘT SỐ MA TRẬN ĐIỂN HÌNH

Phép tịnh tiến



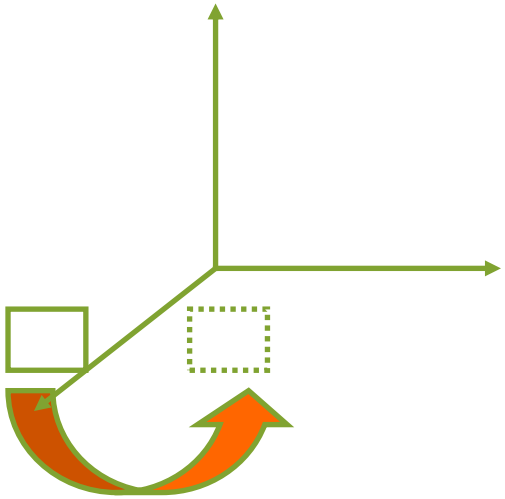
$$\begin{pmatrix} x+t_x \\ y+t_y \\ z+t_z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Thu-phóng theo 3 phương



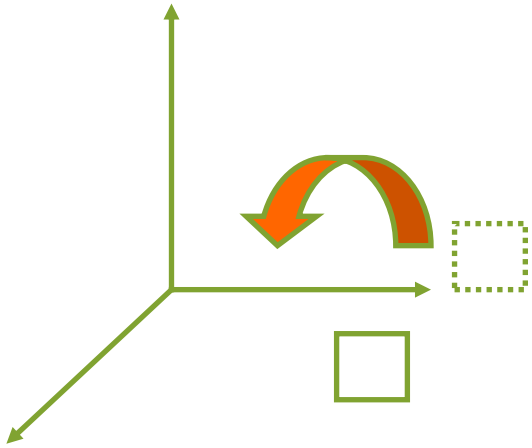
$$\begin{pmatrix} ax \\ by \\ cz \\ 1 \end{pmatrix} = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Quay quanh trục x



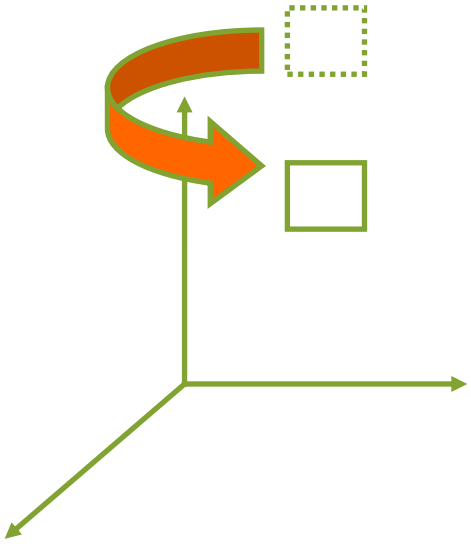
$$\begin{pmatrix} x \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Quay quanh trục y



$$\begin{pmatrix} x' \\ y \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Quay quanh trục z



$$\begin{pmatrix} x' \\ y' \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Ma trận Danevit-Hartenberg

- Đã nghe đến ma trận Danevit-Hartenberg?
- Hãy dùng nó để tìm ma trận định vị của các vật rắn trong robot.

Biến đổi liên tiếp

- Có thể thực hiện một loạt các biến đổi

$$\mathbf{b} = \mathbf{H}_1 \cdot \mathbf{a}$$

$$\mathbf{c} = \mathbf{H}_2 \cdot \mathbf{b}$$

$$\mathbf{d} = \mathbf{H}_3 \cdot \mathbf{c}$$

- Thay thế để có được

$$\mathbf{d} = \mathbf{H}_3 \cdot \mathbf{H}_2 \cdot \mathbf{b}$$

$$\mathbf{d} = \mathbf{H}_3 \cdot \mathbf{H}_2 \cdot \mathbf{H}_1 \cdot \mathbf{a} = \mathbf{H} \,.\, \mathbf{a}$$

với $\mathbf{H} = \mathbf{H}_3 \cdot \mathbf{H}_2 \cdot \mathbf{H}_1$

- Chú ý thứ tự ngược của phép nhân

OpenGL sử dụng ma trận biến đổi đồng nhất

Lớp matrix4x4f

- Là một lớp đối tượng tiện ích trong C++ để thao tác trên các ma trận đồng nhất
- Lấy từ www.codesampler.com/oglsrc/oglsrc_2.htm
- Các thao tác chính: xây dựng ma trận đồng nhất từ các phép biến đổi affine (tịnh tiến, quay, thu phóng…), cộng, trừ và nhân ma trận, đọc vào và ghi ra tệp tin dạng văn bản
- Kết quả có thể được dùng để truyền tham số cho hàm glMultMatrixf() để đặt đối tượng vào vị trí mong muốn

Lớp matrix4x4f

```
class matrix4x4f
{
public:
   float m[16];
   matrix4x4f()  { identity(); }
   matrix4x4f( float m0, float m4, float  m8, float m12, float m1, float m5, float  m9, float m13, float m2, float m6, float m10, float m14, float m3, float m7, float m11, float m15 );

   // Operators...
   matrix4x4f operator + (const matrix4x4f &other);
   matrix4x4f operator - (const matrix4x4f &other);
   matrix4x4f operator * (const matrix4x4f &other);

   matrix4x4f operator * (const float scalar);
```

Lớp matrix4x4f (tiếp)

```
    void identity(void);
    void translate(const vector3f &trans);
    void translate_x(const float &dist);
    void translate_y(const float &dist);
    void translate_z(const float &dist);
    void rotate(const float &angle, vector3f &axis);
    void rotate_x(const float &angle);
    void rotate_y(const float &angle);
    void rotate_z(const float &angle);
    void scale(const vector3f &scale);
    void transformPoint( vector3f *vec );
    void transformVector( vector3f *vec );
};
std::istream & operator >> (std::istream & ss, matrix4x4f & mat);

std::ostream & operator << (std::ostream & ss, const matrix4x4f & mat);
```

Đọc/ghi ma trận matrix4x4f

Ghi ra tệp tin

```
matrix4x4f m;
m.rotate(45, vector3f(1,2,4
ofstream f(“chuy
f << m << endl;
```

Khai báo ma trận. Mặc định là M đơn vị

Thực hiện thao tác trên ma trận

Khai báo biến tệp tin để ghi

Ghi dữ liệu ma trận vào tệp tin

tệp

```
m;
("chuyenvi.txt");
cout << "Matrix m:
cout  << m << endl;
```

Khai báo biến tệp tin để đọc

Đọc dữ liệu ma trận vào biến m

16 phần tử

In ra màn

```
0.771021      0.645108      -0.252818      0      -0.589319
      0.762896      0.265882      0      0.364396      0.0688534
      0.930264      0      0      0      0      1
```

Sử dụng các phép dịch chuyển đơn lẻ

- Thay vì dùng các ma trận đồng nhất, OpenGL cho phép thực hiện các phép biến đổi đơn lẻ, như:
  - glRotate() để quay đối tượng quanh một trục
  - glTranslate() để tịnh tiến đối tượng
  - glScale() để thu phóng đối tượng
  - …

Nội dung

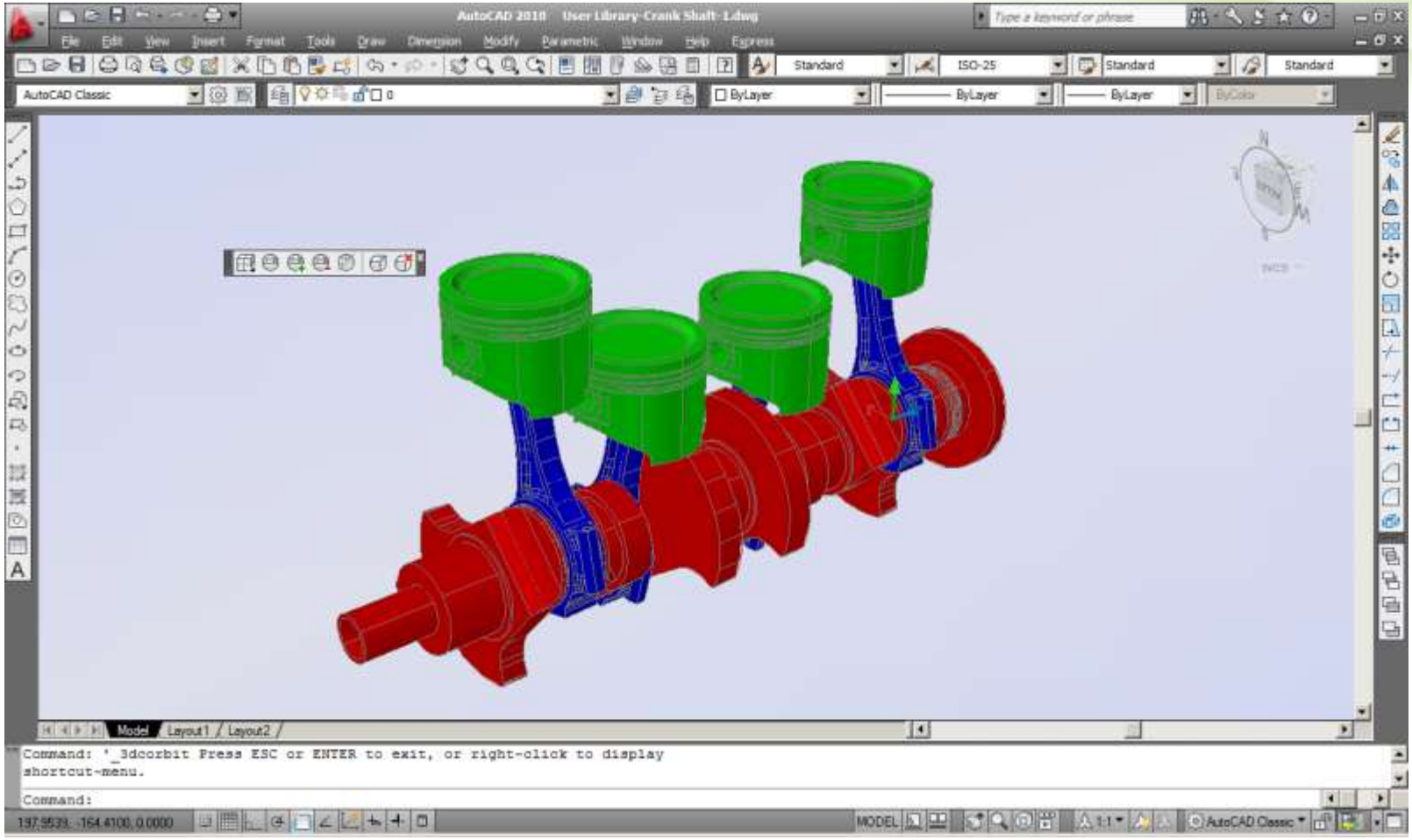
1. Đọc và hiển thị các tệp STL	• Cấu trúc file ASCII STL • Lớp CSTL_File • Sử dụng lớp CSTL_File
2. Sử dụng ma trận đồng nhất trong mô phỏng	• Mô hình hệ mặt trời-trái đất-mặt trăng • Ma trận chuyển vị đồng nhất • Sử dụng các phép dịch chuyển đơn lẻ
3. Mô phỏng chuyển động của robot bằng OpenGL	• Các bước thực hiện • Minh họa: Mô phỏng cơ cấu 4 khâu • Bài tập: Mô phỏng robot Scorbot

Các bước thực hiện

MINH HỌA

MÔ PHỎNG ĐỘNG HỌC THUẬN CƠ CẤU 4 KHÂU BẢN LỀ

Ví dụ: Mô phỏng động học thuận cơ cấu 4 khâu bản lề



Bước 1: Tính toán động học và lưu ma trận chuyển vị

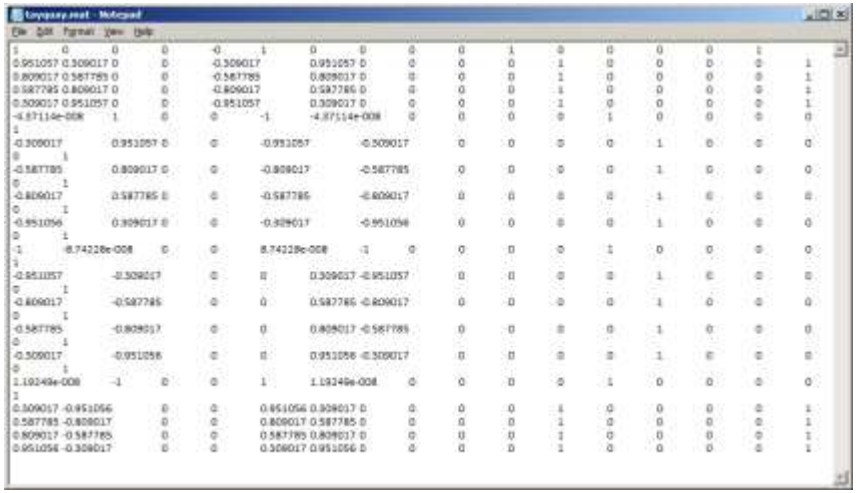
```
const float L1 = 45;
const float L2 = 140;
const float Pi = 3.14159265f;
const int N = 20; // Số điểm chia để tính
const int K = 3; // Số khâu

float phi, psi, xA, yA, yB,
        dphi = 2*Pi / N;

matrix4x4f mat[K], m1, m2;
```

Bước 1: Tính toán (tiếp)

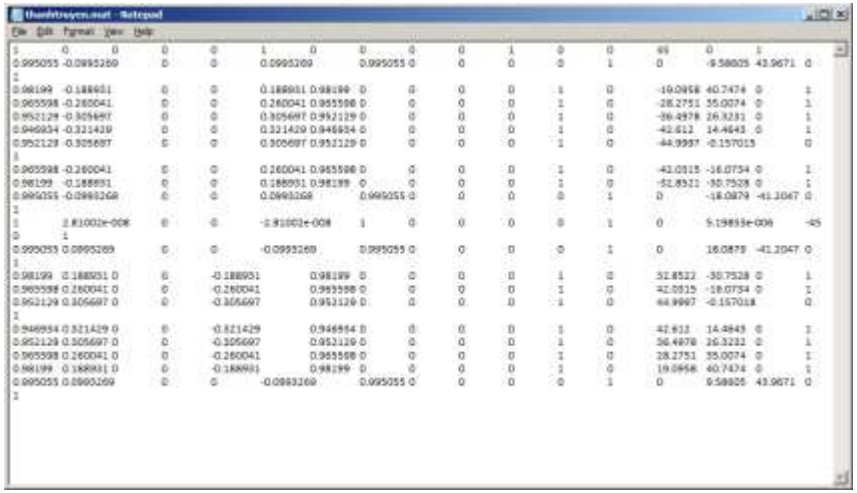
```
for (int i = 0; i < N; i++)
{
    phi = i * dphi;
    xA = -L1*sin(phi);
    yA = L1*cos(phi);
    psi = asin(L1/L2*sin(phi));
    yB = L1*cos(phi)+L2*cos(psi);

    // tay quay
    mat[0].rotate_z(phi);

    // Thanh truyền
    m1.rotate_z(-psi);
    m2.translate(vector3f(xA, yA,
    0.0));
    mat[1] = m1*m2;

    // Con trượt
    mat[2].translate_y(yB);

    // Ghi ra tệp tin
    for (int j = 0; j < K; j++)
        files[j] << mat[j] << endl;
}
```

Bước 1: Ma trận đầu ra

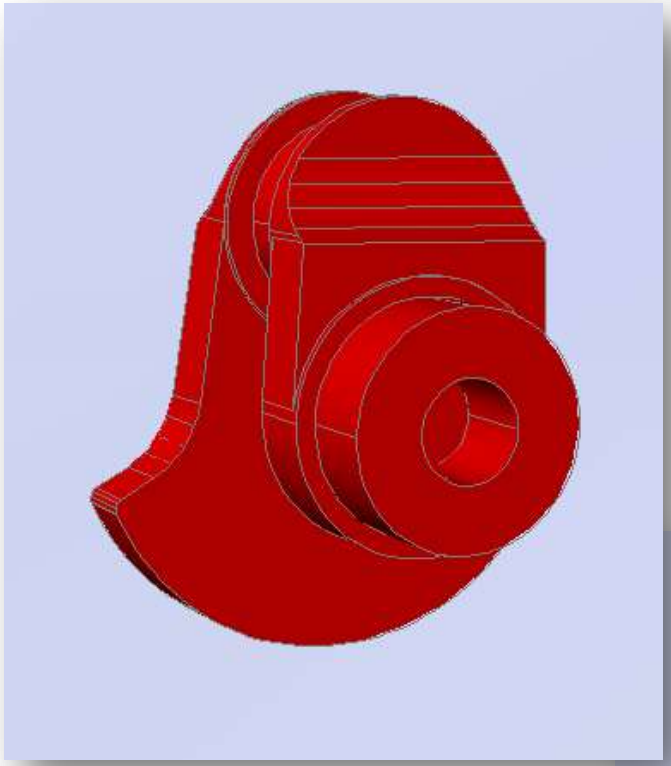
Tayquay.mat

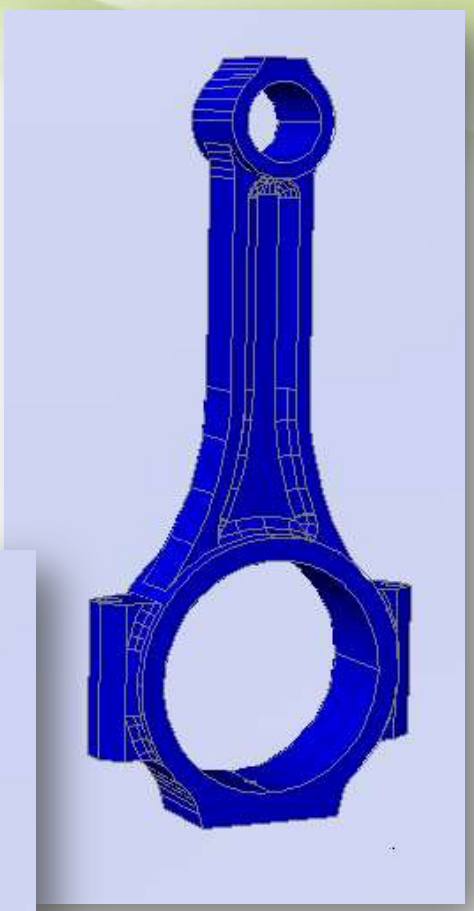


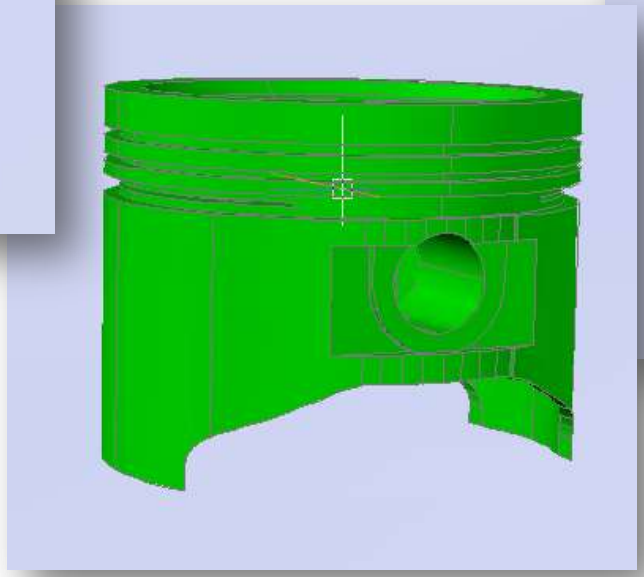
Thanhtruyen.mat



Bước 1: Vẽ đối tượng







Bước 3: Đọc dữ liệu thể hiện và vị trí

```
wstring matnames[K] = {
// Tên các tệp dữ liệu
        L"tayquay.mat",
        L"thanhtruyen.mat",
        L"contruot.mat",};
string stlnames[K] = {
// Tên các tệp dữ liệu
        "tayquay.stl",
        "thanhtruyen.stl",
        "contruot.stl",};
CSTL_File stl[K];
// Tệp STL cho từng khâu
matrixarray mat[K];
// Tệp MAT (ma trận chuyển vị) cho từng khâu
```

```
GLdouble color[3][4] = {
// Màu vẽ các khâu
        {1,0,0,0.5},
        {0,1,0,0.5},
        {0,0,1,0.5},};

Vec3D offset[K] = {
// Dịch vị trí các hình trong file STL
Vec3D(200.0000, 200.0000, 500.0000),
Vec3D(200.0000, 245.0000, 500.0000),
Vec3D(200.0000, 385.0000, 500.0000)
};
```

Bước 3: (tiếp)

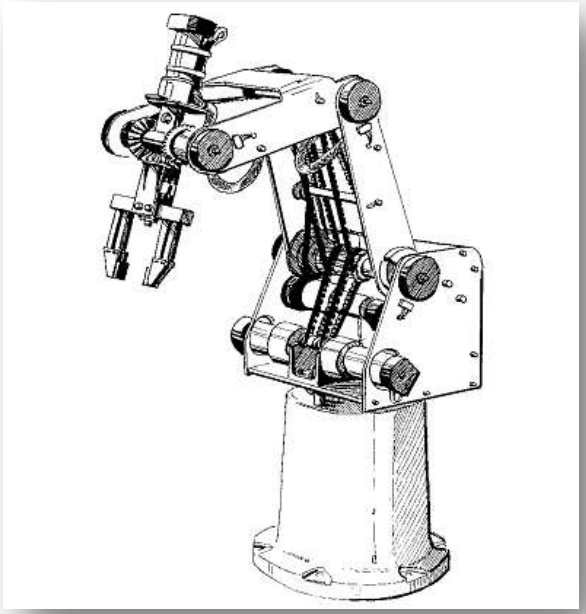
```
for (int j = 0; j < K; j++)
 stl[j].Load(stlnames[j].c_str(),
offset[j]);

for (int j = 0; j < K; j++)
{
   files[j].open(matnames[j].c_
str());
}
matrix4x4f m;
```

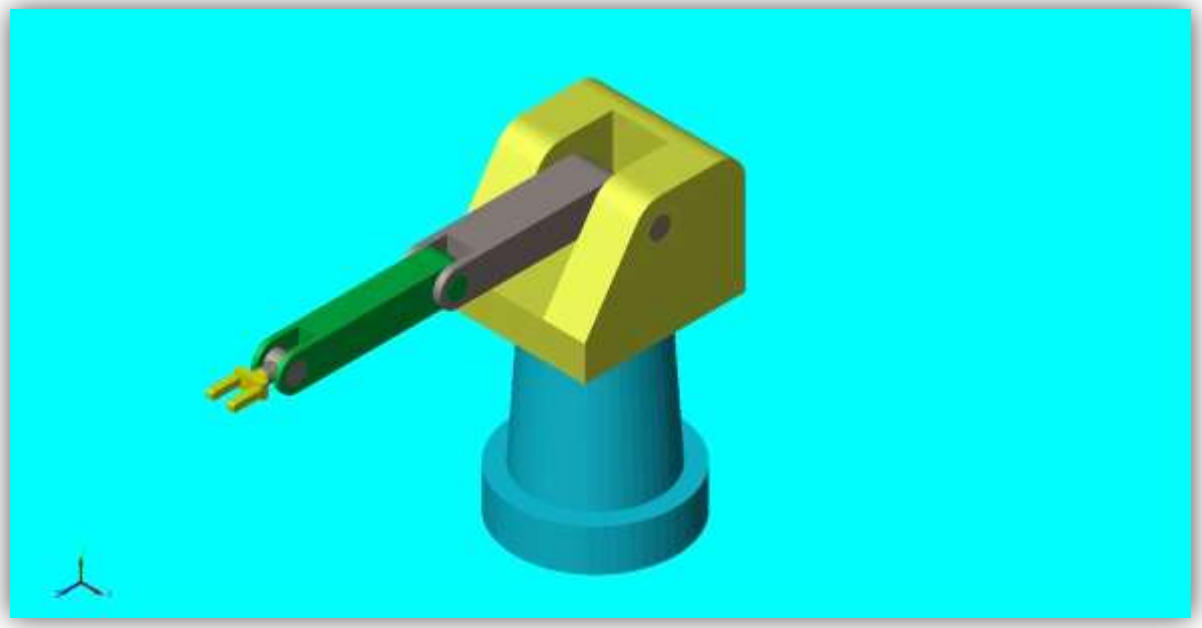
```
while (files[0].good() )
{
  for (int j = 0; j < K; j++)
  {
        files[j] >> m;
        mat[j].push_back(m);
  }
}
```

Bước 4: Thể hiện các khâu tại vị trí yêu cầu

```
for (int j = 0; j < K; j++) // Vẽ các khâu
{
        glPushMatrix();
        {
                glMultMatrixf( mat[j][count].m );
                glColor4dv(color[j]);
                stl[j].Draw(false, true);
        }
        glPopMatrix();
}
count++;
```

Bài tập: Mô phỏng robot Scorbot





Dữ liệu và yêu cầu

- Dữ liệu
  - Bản vẽ Solidworks các khâu
  - Các tệp STL của các khâu
  - Tài liệu hướng dẫn sử dụng robot
- Yêu cầu mô phỏng
  - 1. Mô phỏng động học thuận: cho quy luật chuyển động của từng khâu (lần lượt hoặc đồng thời), ghi dữ liệu vị trí các khâu dưới dạng ma trận chuyển vị, rồi dùng OpenGL vẽ lại.
  - 2. Mô phỏng động học ngược: chọn ra một quỹ đạo của điểm tác động cuối của robot, xác định chuyển động của từng khâu bằng cách giải bài toán động học ngược, ghi dữ liệu vị trí các khâu dưới dạng ma trận chuyển vị, rồi dùng OpenGL vẽ lại.

Có thể mô phỏng các robot khác!

Câu hỏi?