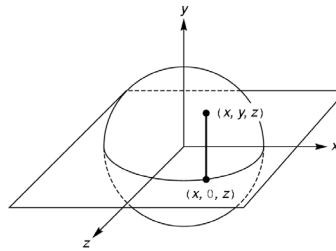


Virtual Trackball

- Use the idle, motion and mouse callbacks in GLUT.
 - `glutMouseMotion()`; `glutMouseFunc()`;
- Mapping the position of a trackball to that of the mouse
 - Assume ball radius = 1
 - project onto plane $y = 0$
 - $(x, y, z) \rightarrow (x, 0, z)$
 - $y = \sqrt{1 - x^2 - z^2}$

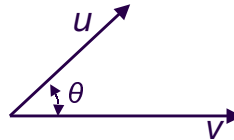


Euclidean Space (Geometry)

- It defines distance between points or length of vectors.
- It is a linear vector space \mathfrak{R}^n or n-tuple scalars :
 - $v = (v_1, v_2, \dots, v_n)$ where $v_i \in \mathfrak{R}$
- vector-vector addition \rightarrow vector
 - $v + u = (v_1, v_2, \dots, v_n) + (u_1, u_2, \dots, u_n) = (v_1 + u_1, v_2 + u_2, \dots, v_n + u_n)$
- scalar-vector multiplication \rightarrow vector
 - $\alpha v = (\alpha v_1, \alpha v_2, \dots, \alpha v_n)$
- Dot or inner product \rightarrow scalar
 - $v \cdot u = (v_1, v_2, \dots, v_n) \cdot (u_1, u_2, \dots, u_n) = v_1 u_1 + v_2 u_2 + \dots + v_n u_n$

Properties of inner product

- o $v \bullet u = u \bullet v$
- o $(\alpha v + \beta u) \bullet w = \alpha v \bullet w + \beta u \bullet w$
- o $u \bullet u > 0$, $u \neq 0$ and $0 \bullet 0 = 0$
 - $u \bullet u = u_1 u_1 + u_2 u_2 + \dots + u_n u_n = u_1^2 + u_2^2 + \dots + u_n^2$
- o Magnitude $|u|$: $u \bullet u = |u|^2$ (from Pythagorean theorem)
 - $|u| = \sqrt{|u \bullet u|} = \sqrt{u_1^2 + u_2^2 + \dots + u_n^2}$
 - $|P-Q| = \sqrt{(P-Q) \bullet (P-Q)}$ (distance between two points)
- o Inner product yields the orientation of 2 vectors
 - $v \bullet u = |u||v| \cos \theta$
 - If $v \bullet u = 0$ and $v \neq 0$ $u \neq 0$
then v and u are orthogonal

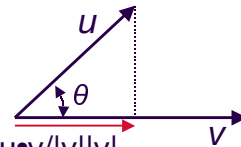


Projections

- o Projection of u onto v (trigonometric solution):
 - $d = |u| \cos \theta$
 - $d = \alpha |v|$
$$\Rightarrow \alpha |v| = |u| \cos \theta$$

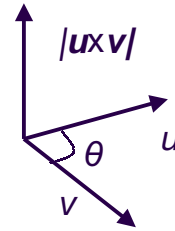
$$\Rightarrow \alpha = |u| \cos \theta / |v| = (|u|/|v|)(u \bullet v / (|u||v|)) = u \bullet v / |v|^2$$

$$\Rightarrow \alpha = u \bullet v / v \bullet v$$
- o Projections are equivalent to the problem of shortest distance from a point to a line or plane.
- o Express vector u as a linear combination of vector v and another vector w orthogonal to v



Cross Product

- Let vectors \mathbf{v} and \mathbf{u} two non-parallel, the cross product $\mathbf{v} \times \mathbf{u}$ produces a new vector \mathbf{n} which is orthogonal to them
 - $\mathbf{n} = \mathbf{v} \times \mathbf{u}$
 - $\mathbf{n} \cdot \mathbf{v} = \mathbf{n} \cdot \mathbf{u} = 0$
- The magnitude of the cross product equals the area of the parallelogram defined by \mathbf{v} and \mathbf{u}
 - $|\mathbf{v} \times \mathbf{u}| = |\mathbf{u}||\mathbf{v}| \sin \theta$
- In 3D space, \mathbf{v} , \mathbf{u} , \mathbf{n} define a **right-handed** coordinate system
- If \mathbf{v} , \mathbf{u} , \mathbf{n} unit vectors $\mathbf{i}, \mathbf{j}, \mathbf{k}$ and $\theta = \pi$ rad, then this is the Cartesian system

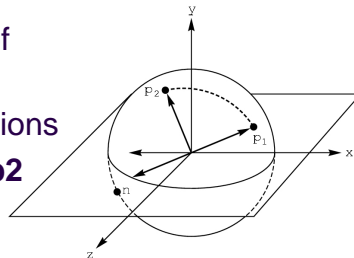


CS1566, Panos K. Chrysanthis – University of Pittsburgh

25

Virtual Trackball (Cont.)

- Determining direction and angle of rotation
 - Let \mathbf{p}_1 and \mathbf{p}_2 two mouse positions
 - Direction of rotation: $\mathbf{n} = \mathbf{p}_1 \times \mathbf{p}_2$
 - Angle: $|\sin \theta| = |\mathbf{n}|$
- For smooth rotation increase angle in small increments: $|\sin \theta| \approx \theta$



$$\mathbf{n} = \mathbf{p}_1 \times \mathbf{p}_2 = \begin{bmatrix} \alpha_2 \beta_3 - \alpha_3 \beta_2 \\ \alpha_3 \beta_1 - \alpha_1 \beta_3 \\ \alpha_1 \beta_2 - \alpha_2 \beta_1 \end{bmatrix}$$

CS1566, Panos K. Chrysanthis – University of Pittsburgh

26

From Window to Unit Hemi-Sphere

```
void trackball_ptov(int x, int y, int width, int height, float v[3]) {
    float d, a;
    /* project x,y onto a hemi-sphere centered within width, height */
    v[0] = (2.0F*x - width) / width;
    v[1] = (height - 2.0F*y) / height;
    /* compute the value on the third axis */
    d = (float) v[0]*v[0] + v[1]*v[1];
    if (d>=1) v[2]=0;
    else v[2]=sqrt(1-d);
    /* Normalize the vector v */
    a = 1.0F / (float) sqrt(v[0]*v[0] + v[1]*v[1] + v[2]*v[2]);
    v[0] *= a; v[1] *= a; v[2] *= a;
}
```

CS1566, Panos K. Chrysanthis – University of Pittsburgh

27

MouseMotion(x,y) Callback

```
float curPos[3], dx, dy, dz, t;
trackball_ptov(x, y, winWidth, winHeight, curPos);

dx = curPos[0] - lastPos[0];
dy = curPos[1] - lastPos[1];
dz = curPos[2] - lastPos[2];

if (dx || dy || dz) {
    axis[0] = lastPos[1]*curPos[2] - lastPos[2]*curPos[1];
    axis[1] = lastPos[2]*curPos[0] - lastPos[0]*curPos[2];
    axis[2] = lastPos[0]*curPos[1] - lastPos[1]*curPos[0];

    t = asin(sqrt(axis[0]*axis[0] + axis[1]*axis[1] + axis[2]*axis[2]));
    angle = 180.0 / PI * t;

    lastPos[0] = curPos[0]; lastPos[1] = curPos[1]; lastPos[2] = curPos[2];
}
```

CS1566, Panos K. Chrysanthis – University of Pittsburgh

28

Implementation in OpenGL

```
bool trackingMouse = false; /* if true, update position */
bool trackballMove = false; /* if true, update rotation matrix */
bool redrawContinue = false; /* if true, idlefunction redraws */

void spinCube() { if (redrawContinue) glutPostRedisplay(); }

void mouseButton(int button, int state, int x, int y) {
    if(button==GLUT_RIGHT_BUTTON) exit();
    if(button==GLUT_LEFT_BUTTON)
        switch(state) {
            case GLUT_DOWN: startMotion(x,y); break;
            case GLUT_UP: stopMotion(x,y); break;
        }
}
```

Implementation in OpenGL

```
void startMotion(int x, int y) {
    trackingMouse = true;  redrawContinue = false;
    startX = x; startY = y;
    curx = x; cury = y;
    trackball_ptov(x, y, winWidth, winHeight, lastPos);
    trackballMove=true;
}

void stopMotion(int x, int y) {
    trackingMouse = false;
    if (startX != x || startY != y) redrawContinue = true;
    else { angle = 0.0F;
           redrawContinue = false;
           trackballMove = false; }
}
```

Display function

```
void display(void) {  
  
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);  
  
    if (trackballMove) {  
        glRotatef(angle, axis[0], axis[1], axis[2]);  
    }  
  
    colorcube();  
  
    glutSwapBuffers();  
}
```

Display function

```
void display(void) {  
    float matrix[4][4];  
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);  
  
    if (trackballMove) { /* view transform -- Recreates first the current  
                        state of the cube and then applies the new rotation */  
        glGetFloatv(GL_MODELVIEW_MATRIX,&(matrix[0][0]));  
        glLoadIdentity();  
        glRotatef(angle, axis[0], axis[1], axis[2]);  
        glMultMatrixf(&(matrix[0][0]));  
    }  
  
    colorcube();  
    glutSwapBuffers();  
}
```