Vstupní operace, identifikace a výběr zobrazovaných objektů, virtuální trackball

Petr Felkel

Katedra počítačové grafiky a interakce, ČVUT FEL
místnost KN:E-413 na Karlově náměstí
E-mail: felkel@fel.cvut.cz

S použitím materiálů Bohuslava Hudce, Jaroslava Sloupa a úprav Vlastimila Havrana

Poslední změna: 3.5.2016

Vstupní operace

- **Fyzické vstupní zařízení**
  myš, pákový ovladač, klávesnice, tablet
  ⇒ fyzická vstupní hodnota
- **Logické vstupní zařízení (LID ... logical input device)**
  abstrakce fyzického vstupního zařízení
  ⇒ logická vstupní hodnota
- **Třídy vstupu:** dělení vstupních operací podle typu logické hodnoty

LOKÁTOR	světové souřadnice (WC) bodu
STROKE	posloupnost bodů (WC)
PICK	identifikace struktury a prvku ve struktuře
CHOICE	číslo alternativy (menu)
VALUATOR	real hodnota (slider)
STRING	řetěz znaků

WC ... world coordinate (souřadnice světového souřadného systému)

DC ... device coordinate (souřadnice zařízení)

Vstup z myši – typy událostí (GLUT)

- **aktivní vstup**
  - **kliknutí** (*click*) glutMouseFunc()
    - událost generovaná při stisku a při uvolnění tlačítka myši.
  - **aktivní přesun – tažení** (*drag*) glutMotionFunc()
    - událost generovaná při změně polohy myši se stisknutým tlačítkem
- **pasivní přesun** (*passive move*) glutPassiveMotionFunc()
  - událost generovaná při změně polohy myši bez stisku tlačítka.
- aplikační program obdrží:
  - pozici kurzoru (DC – souřadnice v okně)
  - identifikaci tlačítka GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON)
  - stav tlačítka po vstupní události (GLUT_UP, GLUT_DOWN)

Použité matice

- O objektová matice (z objektových do světových)
- E^{-1} pohledová matice (ze světových do s. kamery)
- $\frac{1}{w}$ dělení w
- P projektová matice (jediná není afinní)
- V_p transformace záběru (*viewport*) (glGetFloatv(GL_VIEWPORT,…))

Implementace třídy LOCATOR (DC -> WC)

Převádí ze souřadnic na obrazovce do světových (modelových) souřadnic

- Myš vrací 2D polohu kurzoru na obrazovce $\mathbf{s} = (winx, winy)$
- Potřebujeme získat objektovou 3D souřadnici $\mathbf{c}$ bodu, který se do tohoto pixelu $\mathbf{s}$ zobrazil (nebo světovou $O\mathbf{c}$)
- Musíme obrátit proces transformací – zde do modelových
  - Posloupnost transformací bodu $\mathbf{s} = V_p \boxed{\frac{1}{w} P} E^{-1} O\mathbf{c}$
  - Invertujeme na $\mathbf{c} = O^{-1} E \boxed{\frac{1}{w}\ P^{-1}} V_p{}^{-1} \mathbf{s} = \frac{1}{w}\ O^{-1} E P^{-1} V^{-1} \mathbf{s}$

    (násobení $\frac{1}{w}$ lze později, protože matice O^{-1} a E jsou afinní a nemění w)
- S výhodou lze použít **glm::unProject()**
- Chybějící souřadnici $winz$ přečteme z paměti hloubky

Mapping screen ⇒ world coordinates (contd.)

```
glm::vec3 glm::unProject(
        glm::vec3 const & win,                  //        window coordinates
        glm::mat4 const & modelViewMatrix,  // E^{-1}O  or just E^{-1} for world c.
        glm::mat4 const & projMatrix,           // P      Projection
        glm::vec4 const & viewport );           // V_p    NDC to screen space coords
```

- Mapuje souřadnice na obrazovce (win.x, win.y, win.z) do světových či objektových souřadnic (**modelViewMatrix** je $E^{-1}O$, nebo E^{-1})
- **modelViewMatrix** a **projMatrix** jsou aktuální transformační matice
- **viewport** lze získat jedním příkazem glGetFloatv(GL_VIEWPORT), nebo po složkách glutGet(GLUT_WINDOW_WIDTH,…)
  - GLUT_WINDOW_X
  - GLUT_WINDOW_Y
  - GLUT_WINDOW_WIDTH
  - GLUT_WINDOW_HEIGHT Height in pixels of the *current window*.

Mapping screen ⇒ world coordinates (contd.)

picking-stencil-demo

```
GLint winWidth = glutGet(GLUT_WINDOW_WIDTH);
GLint winHeight = glutGet(GLUT_WINDOW_HEIGHT);

// viewport transformation
glm::vec4 viewport = glm::vec4(0, 0, winWidth, winHeight);

GLfloat winZ;          // depth coordinate

// read depth coordinate from depth buffer
glReadPixels(  (GLint)mouseX, (GLint)(winHeight – mouseY - 1), 1, 1,
     GL_DEPTH_COMPONENT, GL_FLOAT, &winZ );

 // window coordinates of the mouse click
glm::vec3 winCoords = glm::vec3(mouseX, winHeight - mouseY - 1, winZ);

glm::mat4 modelMatrix = …;     // modeling transformation matrix – identity for world coords pick
glm::mat4 viewMatrix = glm::lookAt( … );              // view transformation matrix
glm::mat4 modelView = modelMatrix * viewMatrix;
glm::mat4 projection = glm::perspective( … );         // projection transformation  matrix

// map the specified window coordinates into world/object coordinates
glm::vec3 objCoords = glm::unProject(
     winCoords, modelView, projection, viewport);
```

Window coords

Implementace třídy LOKÁTOR v glm

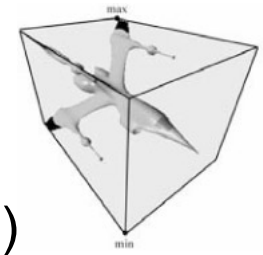
```
template <typename T, typename U>
GLM_FUNC_QUALIFIER detail::tvec3<T> unProject
(
	detail::tvec3<T> const & win,
	detail::tmat4x4<T> const & model,
	detail::tmat4x4<T> const & proj,
	detail::tvec4<U> const & viewport
)
{
	detail::tvec4<T> tmp = detail::tvec4<T>(win, T(1));
	tmp.x = (tmp.x - T(viewport[0])) / T(viewport[2]);
	tmp.y = (tmp.y - T(viewport[1])) / T(viewport[3]);
	tmp = tmp * T(2) - T(1);

	detail::tmat4x4<T> inverse = glm::inverse(proj * model);
	detail::tvec4<T> obj = inverse * tmp;

	obj /= obj.w;		// dělení w, lze později, protože
				// matice jsou afinní
	return detail::tvec3<T>(obj);
}
```

Implementace třídy LOKÁTOR pro float

```
vec3 unProject
(
     vec3 const & win,        // window coords
     mat4 const & modelView, // modelView nebo jen View
     mat4 const & proj,
     vec4 const & viewport   // [x,y,w,h]
)
{    // win -> NDC
     vec4 tmp = vec4(win, 1.0);              //win.z is 0..1
     tmp.x = (tmp.x - viewport[0]) / viewport[2]; //to 0..1
     tmp.y = (tmp.y - viewport[1]) / viewport[3]; //to 0..1
     tmp = tmp * 2.0 - 1.0;  // NDC coords          to -1..1

     mat4 inverse = glm::inverse(proj * modelView);
     vec4 obj = inverse * tmp;

     obj /= obj.w;        // dělení w, lze nakonec, protože
                          // matice modelView je afinní
     return vec3(obj);    // objektové souřadnice
}
```

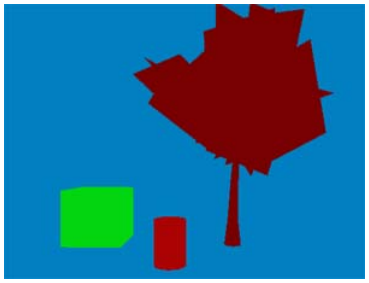
Možné implementace výběru
Selection / picking objektů

Součástí interaktivních aplikací bývá výběr jednotlivých objektů (pick), nebo výběr oblasti (select)



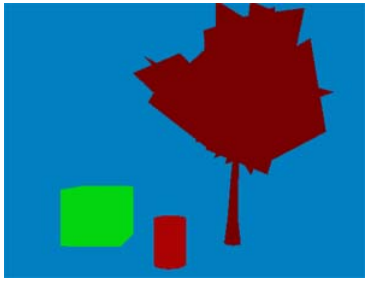
Možné implementace výběru objektů

1. **ohraničující tělesa** (bounding boxes)
   (test, zda bod leží v ohraničujícím tělese)
2. test zda bod náleží grafickému primitivu ve scéně
   ⇒ **traverzování struktury scény**
3. nastavení malého pohledového kvádru se středem v místě kurzoru myši ⇒ objekty blízko kurzoru se tam vykreslí (*OpenGL ≤ 3.0 GL_SELECT mode - deprecated*)
4. označení objektů různými barvami (*lze implementovat v OpenGL*)
   barva pixelu ⇒ odpovídá ID objektu
5. pomocí paměti šablon (*OpenGL*)
   převádí hodnotu v šabloně na
   ⇒ jméno objektu (ID)



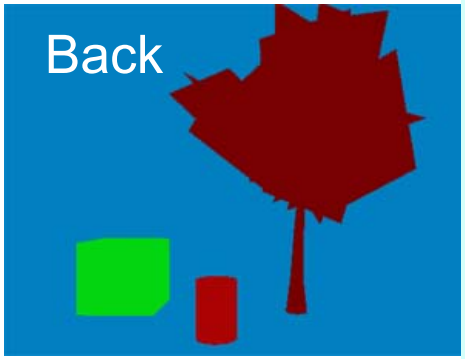
4. Implementace třídy PICK pomocí zadní barevné roviny *(back buffer)*

- Místo barev „vykreslíme“ ID objektů do zadní obrazové paměti *(back buffer)*
- Obrázek nezobrazíme (neprovedeme glutSwapBuffers)
- Při výběru přečteme ID objektu pixelu na pozici myši příkazem glReadPixels()
- Dvě velmi podobné varianty
  a) Kreslíme stále do dvou pamětí – dopředu obrázek, dozadu ID
  b) Kreslíme ID pouze ve chvíli, kdy dojde ke kliknutí myši



Při více výběrech ve statické scéně lze zadní buffer s ID využít opakovaně a nemusí se překreslovat

4a) PICK – kreslíme zároveň do dvou obrazových pamětí,




```
void draw() {  // do dvou obrazových pamětí
   DrawBuffer( GL_FRONT );
   glClear(GL_COLOR_BUFFER_BIT | …);
   drawScene (); //obrázek do přední paměti
   glDrawBuffer( GL_BACK );
   glClear(GL_COLOR_BUFFER_BIT | …);
   drawSceneWithColorIDs();  // ID do zadní
}
// po kliknutí myši přečteme ID
void processHit (int x, int y) {
   GLubyte pixel[3];
   GLfloat viewport[4];

   glGet(GL_VIEWPORT, viewport );
   glReadBuffer( GL_BACK ); // ID ze zadní
   glReadPixels( x, viewport[3]-y-1, 1, 1,
         GL_RGB, GL_UNSIGNED_BYTE, pixel );
} // pixel obsahuje ID
```

Na některých starších kartách chybně (zaokrouhlí jinak float)

4b) PICK – ID vykreslíme jen v okamžiku kliknutí

- ID vykreslíme a čteme pouze po kliknutí myši (1x)
- Funkce `do_picking()` vykreslí ID objektů a přečte ID v pixelu na pozici myši

```
void mouseCallback(int button, int state, int mx, int my)
{
  // do picking only on mouse down
  if(state == GLUT_DOWN)
  {
    doPicking(button, x, height - 1 - y); // recalculate y,
                                          // as glut has origin in upper left corner
    glutPostRedisplay();                  // vynecháme  glutSwapBuffers!!
  }
}
```

Picking v OpenGL, část C++ 1/2

```
void doPicking (int button, int winX, int winY)  {

  glClearColor(0.0f, 0.0f, 0.0f, 0.0f);      // 0 = background
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glUseProgram(pickProgram);  // encoding object ID into fragment color

  for(int b = 0; b < numberOfObjects; ++b)  {   // encode ID's to colors

    glUniform1f(id_loc, (float)b / 255);  // set ID as a float (in FS)
    …
    glBindVertexArray( objectDataVAO );    // object's vertices
    glDrawArrays(GL_TRIANGLES, 0, objectSize );
    glBindVertexArray( 0 );

    glFinish();  // wait for all primitives are drawn
  }
  …
```

Picking v OpenGL, část C++ 2/2

```
…
unsigned char pixel[4]; // variable to store pixel value

// reading of the pixel with ID under the mouse pointer
glReadPixels(winX, winY, 1, 1, GL_RGBA, GL_UNSIGNED_BYTE,  pixel);


// buffer was cleared with zeros
if(pixel[1] == 0) // background has pixel[1] == 0 (see fragment shader)
  std::cout << "clicked on background" << std::endl;
else // click on object
    std::cout << "clicked on object " << (int)pixel[0]
            << " in depth " << (float)pixel[2] * MAX_DEPTH / 255
            << std::endl;
}
```

Picking v OpenGL, část vertex shader

```
// vertex shader for picking
#version 130
uniform mat4 M;            // modeling transformation matrix
uniform mat4 P;            // projection transformation matrix
in vec3 position;          // vertex position
smooth out float depth;    // vertex depth/distance in camera space

void main()
{
  vec4 pos4 = M * vec4(position, 1);
  gl_Position = P * pos4;
  depth = pos4.z;                    // vertex dist in camera space
}
```

Picking v OpenGL, část fragment shader

```
// fragment shader for picking
#version 130
uniform float id;
smooth in float depth;
out vec4 outId;

void main()
{
   // we can encode anything into the fragment color,
   // we use ID and depth here
   outId = vec4(id, 1, -depth / MAX_DEPTH, 0);
}
```

ID

Free place for any other attribute

picking-demo

5. Picking using the stencil buffer

principle:

- stencil buffer contains for each pixel one byte (8-bits)
- during the drawing of the scene we write objects ids into the stencil buffer (0 → background, 1 – 255 → object IDs)
- mouse click → read stencil buffer value for a pixel under the mouse click → object ID

limitations of this technique:

- maximum number of pickable objects in the scene is 255 (object IDs 1 – 255, 0 is reserved for the background), because of the 8 bits per pixel in the stencil buffer
- to get a hit location in world space read the depth buffer and unProject it

Picking using the stencil buffer (contd.)

```
void doPicking(int button, int winX, int winY)  {

  glClearStencil(0); // this is the default value
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT );
 ...
  // enable stencil operations
  glEnable(GL_STENCIL_TEST);
  // if the stencil test and depth test are passed than value in the stencil
  // buffer is replaced with the object ID (byte 1..255)
  glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);

  for(int i = 0; i < numObjects; i++) {  // draw objects with IDs, ID=i+1
     // stencil test always passes
     // reference value for stencil test is set to be object ID (b+1)
     // -> this value is used to update stencil buffer contents
     glStencilFunc(GL_ALWAYS, i + 1, -1);  // set object ID
     drawObject(i);
  }
```

Is this KEEP important?
Why not **REPLACE**?

Picking-stencil-demo

Picking using the stencil buffer (contd.)

```
unsigned char pixelID; // stores value from the stencil buffer (byte)

// read ID's from the stencil buffer for one pixel under the mouse cursor

glReadPixels(winX, winY, 1, 1,
             GL_STENCIL_INDEX, GL_UNSIGNED_BYTE, &pixelID);

// disable stencil test
glDisable(GL_STENCIL_TEST);

// the buffer was cleared to zeros
if(pixelID == 0) {                          // background was clicked
  std::cout << "clicked on background" << std::endl;
}
else {                                      // object was clicked
  std::cout << "clicked on object with ID: " << (int)pixelID << std::endl;
}
}    // Note: ReadPixels can read color, depth, or stencil buffers
```

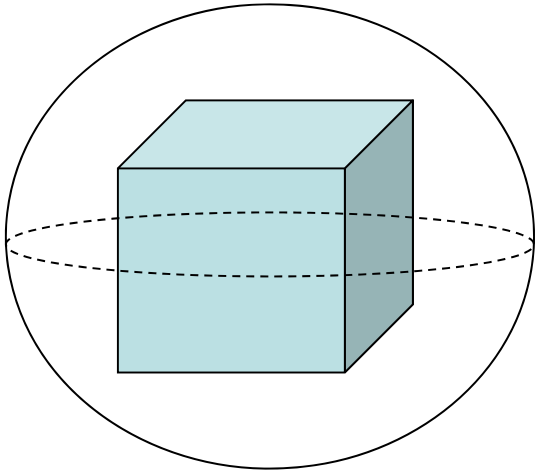
Virtual trackball

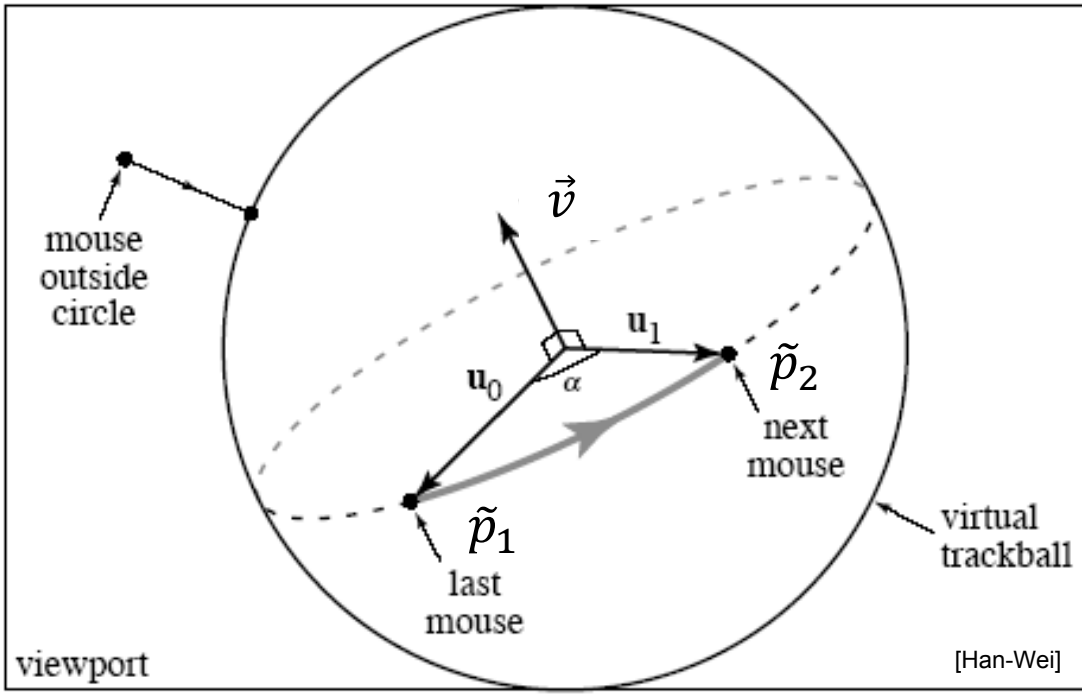
- Mapuje 2D plochu na 3D povrch polokoule
- Najde obecnou osu a úhel => nenastane gimbal lock!!!

- Kvaterniony
  - vhodné pro ukládání, sčítání, …
  - zatím se obejdeme bez nich

Virtual trackball - princip

- Představa: objekt obalen koulí se středem v počátku $\tilde{o}$ soustavy souřadnic objektu $\vec{\boldsymbol{o}}^t$ a otáčí se vůči kameře $\vec{\boldsymbol{e}}^t$ resp. obrazovce





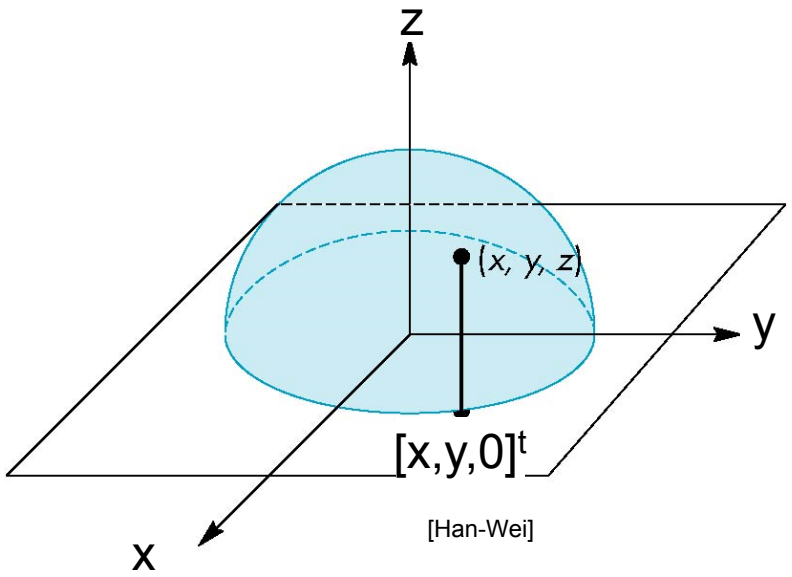
[Han-Wei]

- Dva body $\tilde{p}_1$ a $\tilde{p}_2$ definují začátek a konec otáčení

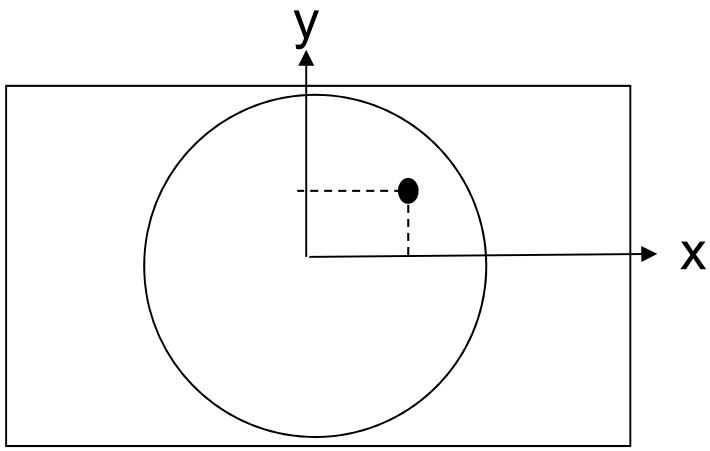
=> Úhel α a vektor obecné osy otáčení $\vec{v}$

Virtual trackball – princip získání Z

- Na záběr položíme polokouli
- Polokoule se na obrazovku promítne jako kružnice
- Myší zadáme x a y
- Promítneme polohu myši na povrch koule
  (pro x a y získáme z)

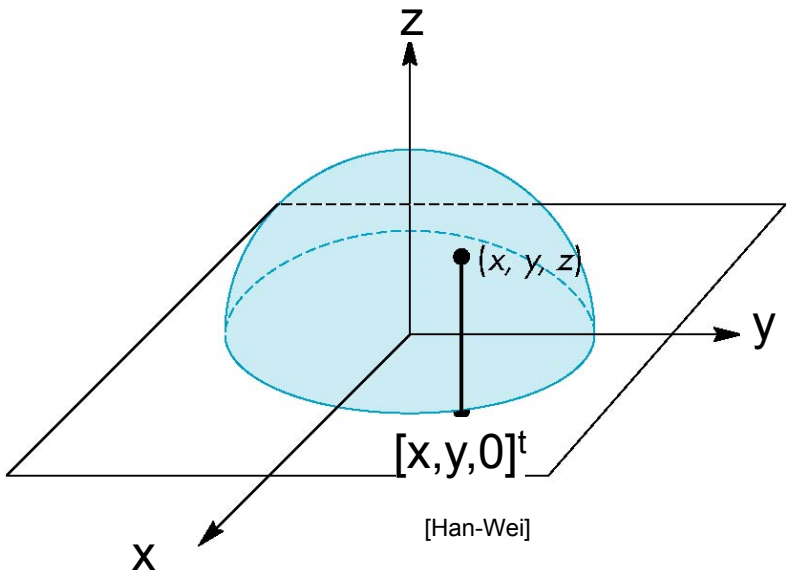


[Han-Wei]



Virtual trackball – princip získání Z

Promítneme polohu myši na povrch koule
(pro x a y získáme z)



[Han-Wei]

- Koule o poloměru r
- Bod na kouli se promítá na rovinu $z = 0$

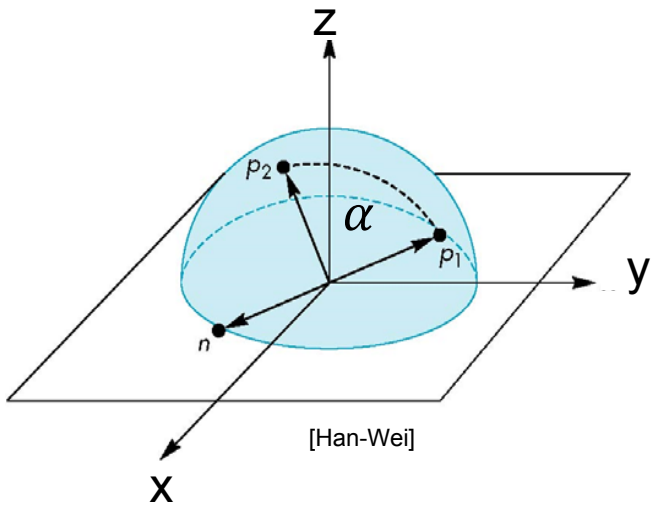
$$[x \quad y \quad z]^t \Rightarrow [x \quad y \quad 0]^t$$

- Pro známé $[x \quad y \quad 0]^t$ vypočteme z

$$z = \sqrt{(r^2 - x^2 - y^2)}$$

Virtual trackball - princip

- Uložíme předchozí polohu myši $\tilde{p}_1$ a sledujeme aktuální $\tilde{p}_2$
- Normála $\vec{n}$ roviny $(\tilde{p}_1, \tilde{p}_2, \tilde{o})$ je osa otáčení okolo $\tilde{o}$
- Rotujeme objekt dle osy $\vec{n}$ o správný úhel α
  - Původní transformace O
  - Otočení $R(\alpha)$
  - Nová transformace $O' \leftarrow R(\alpha)O$



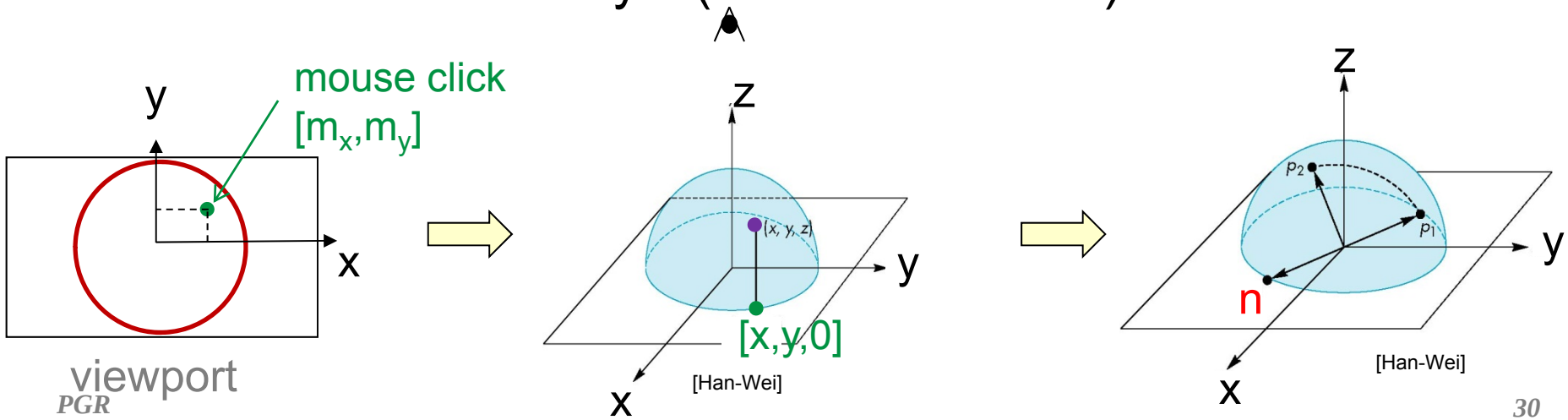
[Han-Wei]

$$\vec{\boldsymbol{o}}^t = \vec{\boldsymbol{w}}^t O \Rightarrow \vec{\boldsymbol{w}}^t R(\alpha) O = \vec{\boldsymbol{w}}^t O'$$

- Před interakcí si uchováme původní modelovou matici O
- Během interakce k ní akumulujeme nové otočení
- Po dokončení interakce ji aktualizujeme dle posledního α

Virtual trackball

1. Detekuj stisk myši a ulož si 3D souřadnice -> P1
2. Sleduj pohyb myši (drag) -> P2
   a) Promítni 2D body P1, P2 na kouli
   b) Urči osu rotace od P1 k P2 (normálu $\vec{n}$ roviny) a úhel α
   c) Přinásob pootočení k rotaci trackballu před stiskem myši v P1
   d) Překresli scénu, jdi na a)
3. Ukonči sledování myši (uvolnění tlačítka)



Virtual trackball s drobnými úhly

1. Detekuj stisk myši a ulož si 3D souřadnice -> P1
2. Sleduj pohyb myši (drag) -> P2
   a) Promítni 2D body P1, P2 na kouli
   b) Urči osu rotace od P1 k P2 (normálu $\vec{n}$ roviny) a úhel α
   c) Přinásob pootočení k rotaci trackballu před stiskem myši v P1
   d) Nastav P1 = P2 (otáčení o drobné úhly)
   e) Překresli scénu, jdi na a)
3. Ukonči sledování myši (uvolnění tlačítka)

Virtual trackball - implementace

- Proměnná `trackballRotationMatrix` obsahuje celkovou rotaci trackballu (poskládanou z malých rotací od P1 k P2)
- Při vykreslování se skládá s aktuální modelovou maticí (násobí ji zleva) $O' \leftarrow R(\alpha)O$

```
glm::mat4 newModel = trackballRotationMatrix * modelMatrix;
```

a jako modelová matice používá se ta složená

```
// složená modelová matice
glUniformMatrix4fv( modelMatrixLocation, 1, GL_FALSE,
                    glm::value_ptr(newModel));
drawModel();
```

- Viz demos/trackball-demo a pgr-Framework/src/trackball.cpp

Virtual trackball - implementace

1. Detekce stisku myši a uložení výchozích souřadnic P1

```
//mouse button pressed within a window or released
void mouseCb(int button, int state, int x, int y) {

  if(button == GLUT_LEFT_BUTTON) {
    if(state == GLUT_DOWN) {
      startGrabX = x;                  // starting point P1
      startGrabY = y;                  // updated in mouseMotionCb
      glutMotionFunc(mouseMotionCb); // 2. start mouse tracking
      return;
    }
    else {  // GLUT_UP
      glutMotionFunc(NULL);          // 3. stop mouse tracking
    }
  }
}
```

Virtual trackball - implementace

2. Sleduj pohyb myši -> P2

```
// mouse motion within the window with any button pressed (drag)
void mouseMotionCb(int x, int y) {
  endGrabX = x; // point P2
  endGrabY = y;

  if(startGrabX != endGrabX || startGrabY != endGrabY) {
    /* get rotation increment from trackball */
    trackball.addRotation(startGrabX, startGrabY, endGrabX,//2abc)
                          endGrabY, winWidth, winHeight);
    /* build rotation matrix from trackball rotation */
    trackball.getRotationMatrix(trackballRotationMatrix);
    startGrabX = endGrabX;  // move point P1 to current P2
    startGrabY = endGrabY;
    glutPostRedisplay(); // 2d)
  }
}
```

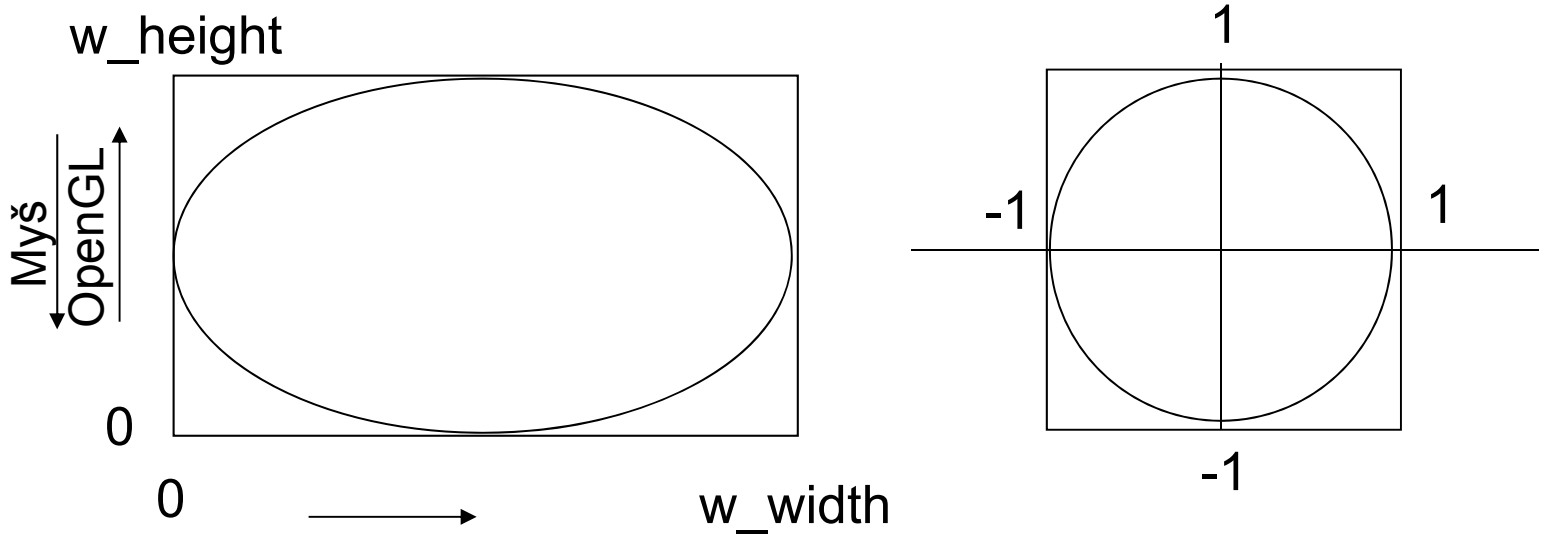
Virtual trackball - implementace

2abc) Přidání pootočení k celkové rotaci trackballu

```
void CClassicTrackball::addRotation(
                int startPointX, int startPointY,
                int endPointX,   int endPointY,
                int winWidth,    int winHeight ) {
 float endX, endY, startX, startY;
 if(startPointX == endPointX && startPointY == endPointY)
    return;   // no move => no rotation
 mapScreenCoords( startX, startY, startPointX, startPointY,// 2a)
                  winWidth, winHeight); //P1:(0..w, h..0)=>(-1..1,-1..1)
 mapScreenCoords( endX, endY, endPointX, endPointY,        // 2a)
                  winWidth, winHeight); //P2: dtto
 glm::mat4 newRotation; // rotation increment
 computeRotation(newRotation, startX, startY, endX, endY); // 2b)

 // trackball rotation = rotation increment * previous rotation
 _rotationMatrix = newRotation * _rotationMatrix;          // 2c)
}
```

Virtual trackball – mapování na kouli

2a) Mapování souřadnice myši na normalizovanou kouli



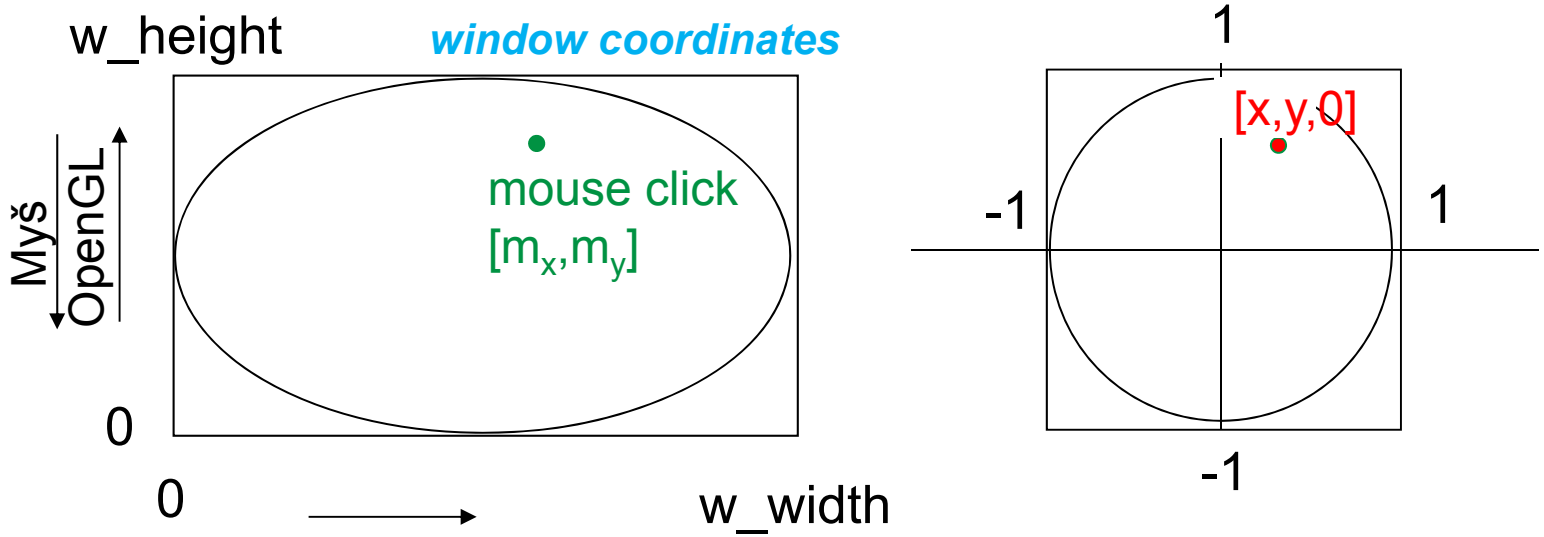
Převod intervalu ⟨0, w_width⟩ na rozsah ⟨-1, 1⟩

x' = – 1.0 + (screen_x / w_width) * 2.0

y' = 1.0 – (screen_y / w_height) * 2.0

Virtual trackball - implementace

2a) Mapování souřadnice myši na plochu koule




```
void CTrackball::mapScreenCoords(float& outX, float& outY,
        int screenX, int screenY, int winWidth, int winHeight){
  outX = -1.0f + 2.0f * screenX / winWidth;
  outY =  1.0f - 2.0f * screenY / winHeight;
}
```

Virtual trackball - implementace

2b) Výpočet matice pootočení trackballu $R(\alpha)$

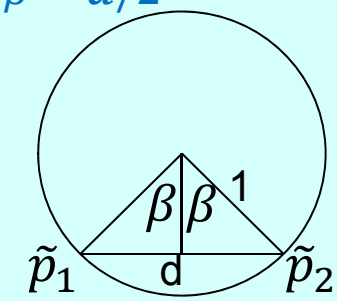
```
void CClassicTrackball::computeRotation(glm::mat4& rotation,
  float startPointX, float startPointY, float endPointX, float
  endPointY)
{
  if(startPointX == endPointX && startPointY == endPointY) {
     rotation = glm::mat4(1.0); // Zero rotation
     return;
  }
  glm::vec3 axis;
  float angle;

  computeRotationAxisAndAngle(axis, angle,
                startPointX, startPointY, endPointX, endPointY);

  rotation = glm::rotate(glm::mat4(1.0f), angle, axis);
}
```

Virtual trackball - implementace

2b) Výpočet osy otáčení a úhlu α

```
#define TRACKBALLSIZE  (0.8f)
```

```
void CTrackball::computeRotationAxisAndAngle(
        glm::vec3& axis, float& angle, float startPointX,
        float startPointY, float endPointX, float endPointY)
{  // Compute z-coordinates for projection of P1, P2 to sphere
  glm::vec3 p1(startPointX, startPointY,
        projectToSphere(TRACKBALLSIZE, startPointX, startPointY));
  glm::vec3 p2(endPointX, endPointY,
        projectToSphere(TRACKBALLSIZE, endPointX, endPointY));
  axis = glm::normalize(glm::cross(p1, p2));   // AXIS
  // ALGLE to rotate around that axis.
  glm::vec3 d = p1 - p2; // chord (tětiva), t=sin(β), β = α/2
  double t = glm::length(d) / (2.0 * TRACKBALLSIZE);
  //* Avoid problems with out-of-control values...s
  if(t > 1.0)     t = 1.0;
  if(t < -1.0)    t = -1.0;
  angle = float(RADTODEG(2.0f * asin(t)));   // ANGLE
}                                            // (in degrees)
```

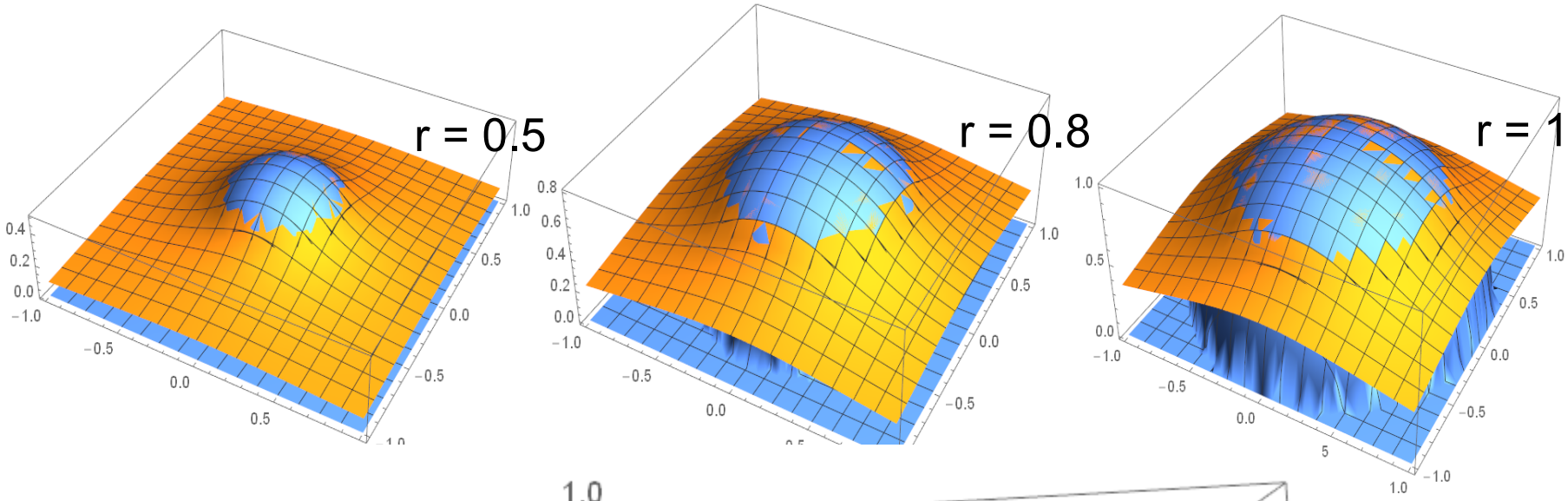


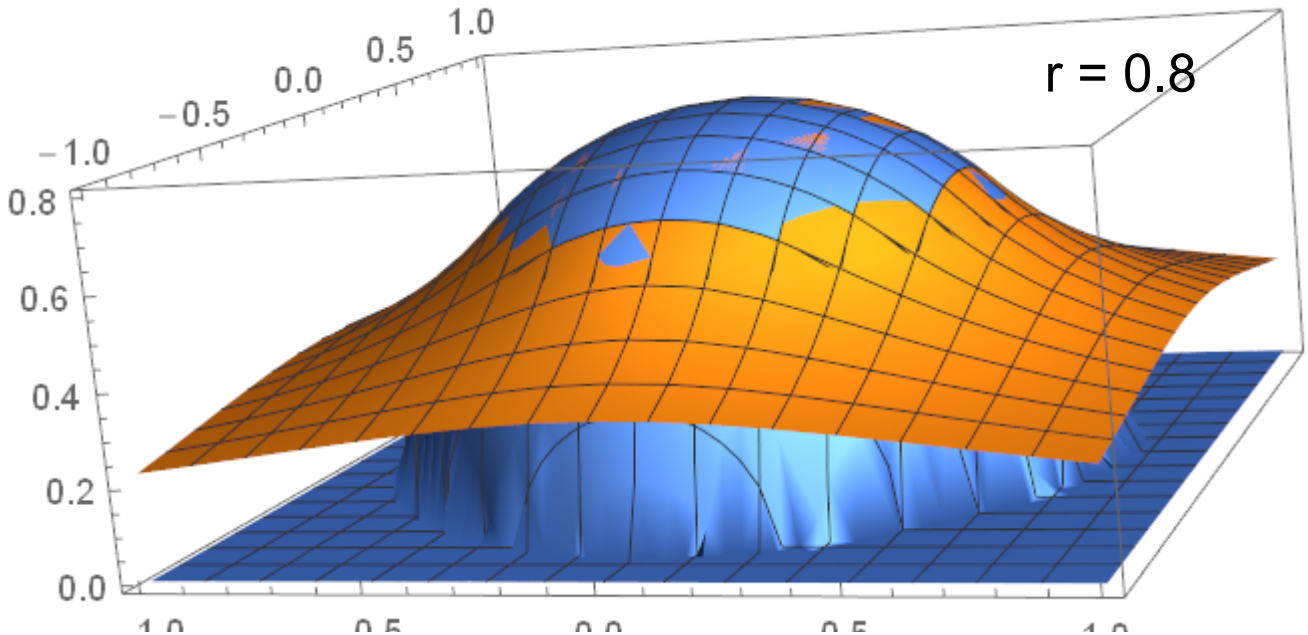
Virtual trackball - implementace

2a) Projekce 2D bodu na kouli (uvnitř) či hyperbolu (vně)

```
float CTrackball::projectToSphere(float radius,float x,float y)
{
  double d, t, z;

  d = sqrt(x*x + y*y);
  if(d < radius * 0.70710678118654752440) {     // Inside sphere
    z = sqrt(radius*radius - d*d);
  }
  else {                                        // On hyperbola
    t = radius / 1.41421356237309504880;
    z = t*t / d;
  }

  return (float)z;
}
```

Projekce na kouli a hyperplochu





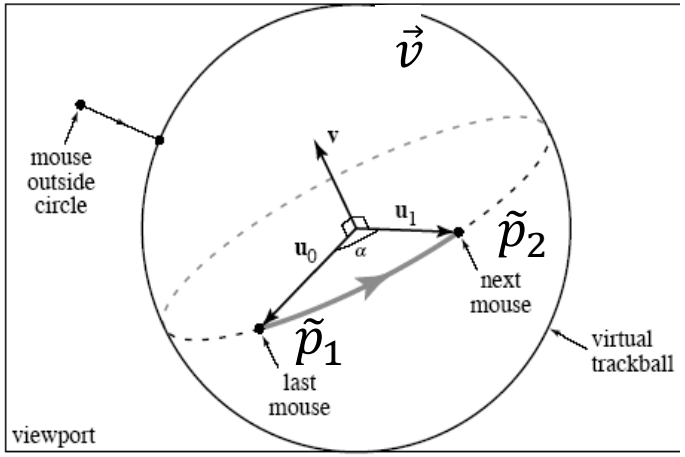
Shoemake: Cross section of the mapping surface

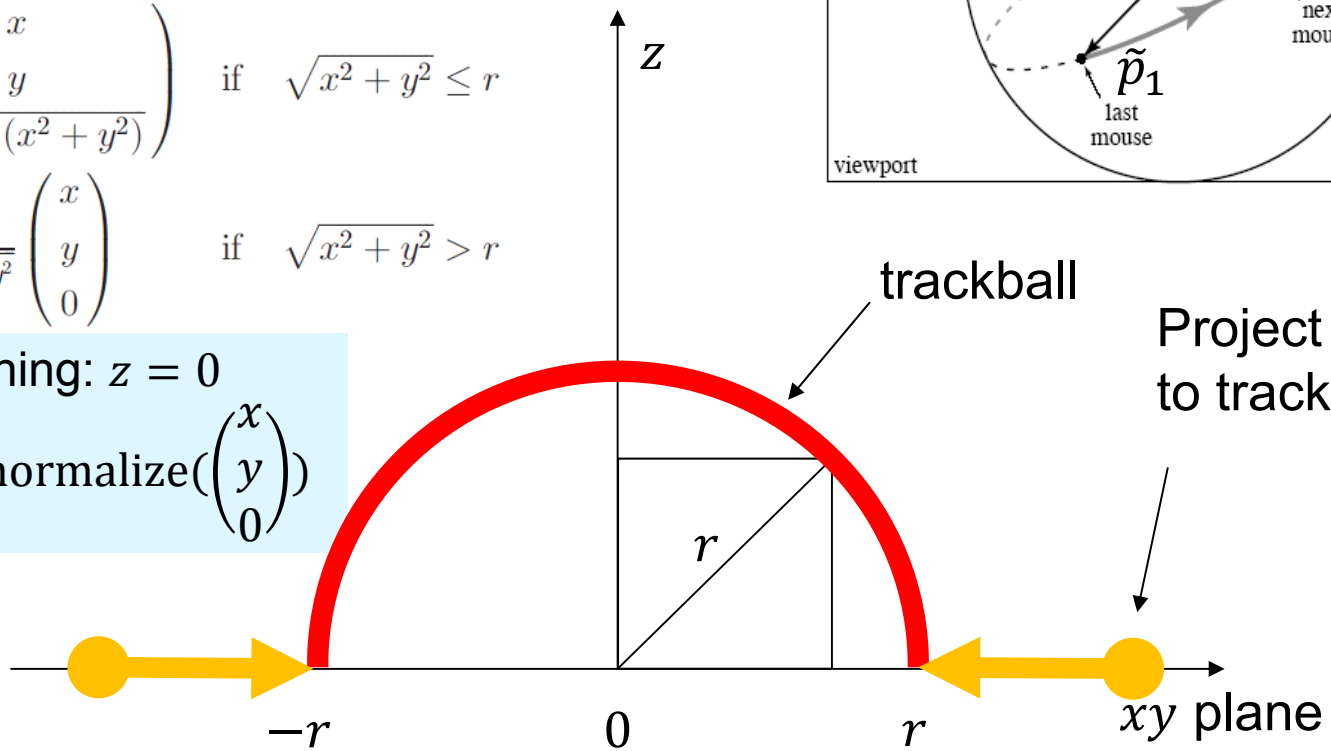
Maps points outside to the circle

$$= \begin{cases} \begin{pmatrix} x \\ y \\ \sqrt{r^2 - (x^2+y^2)} \end{pmatrix} & \text{if} \quad \sqrt{x^2+y^2} \le r \\ \frac{r}{\sqrt{x^2+y^2}} \begin{pmatrix} x \\ y \\ 0 \end{pmatrix} & \text{if} \quad \sqrt{x^2+y^2} > r \end{cases}$$

Meaning: $z = 0$

$$r * \text{normalize}\left(\begin{pmatrix} x \\ y \\ 0 \end{pmatrix}\right)$$





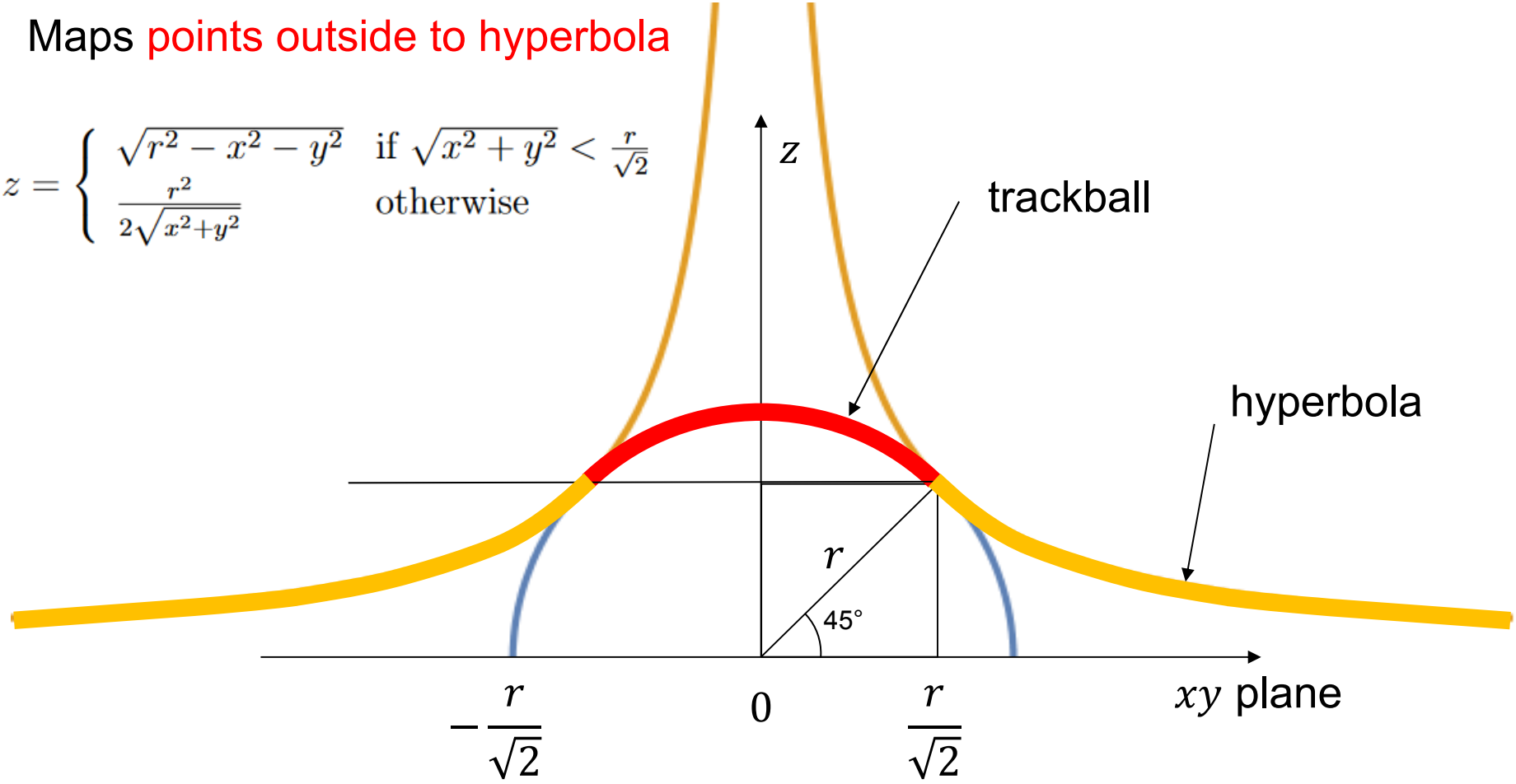
Project the point to trackball

[Henriksen04]

Bell: Cross section of the mapping surface

Maps points outside to hyperbola

$$z = \begin{cases} \sqrt{r^2 - x^2 - y^2} & \text{if } \sqrt{x^2+y^2} < \frac{r}{\sqrt{2}} \\ \frac{r^2}{2\sqrt{x^2+y^2}} & \text{otherwise} \end{cases}$$



[Henriksen04]

References to virtual trackball

- Roger Crawfis: “Implementing a Virtual Trackball or Examiner Viewer”, http://www.cse.ohio-state.edu/~crawfis/cis781/Slides/VirtualTrackball.html
  - pro starou verzi OpenGL, kterou se zde neučíme
  - Mírně odlišný postup – po celou dobu interakce se vztahuje k prvnímu bodu P1
- Gavin Bell: Implementation of a virtual trackball https://github.com/danping/CoSLAM/blob/master/src/gui/trackball.cpp a .h
  - Zde uvedená verze trackballu
- [Han-Wei] Han-Wei Shen. Quaternion and Virtual Trackball. CSE 781 Introduction to 3D Image Generation, 2007 http://www.cs.sunysb.edu/~mueller/teaching/cse564/trackball.ppt