

PV112 – Programování grafických aplikací

4. přednáška – Transformace –
pokračování, Display listy, Vertex
Arrays

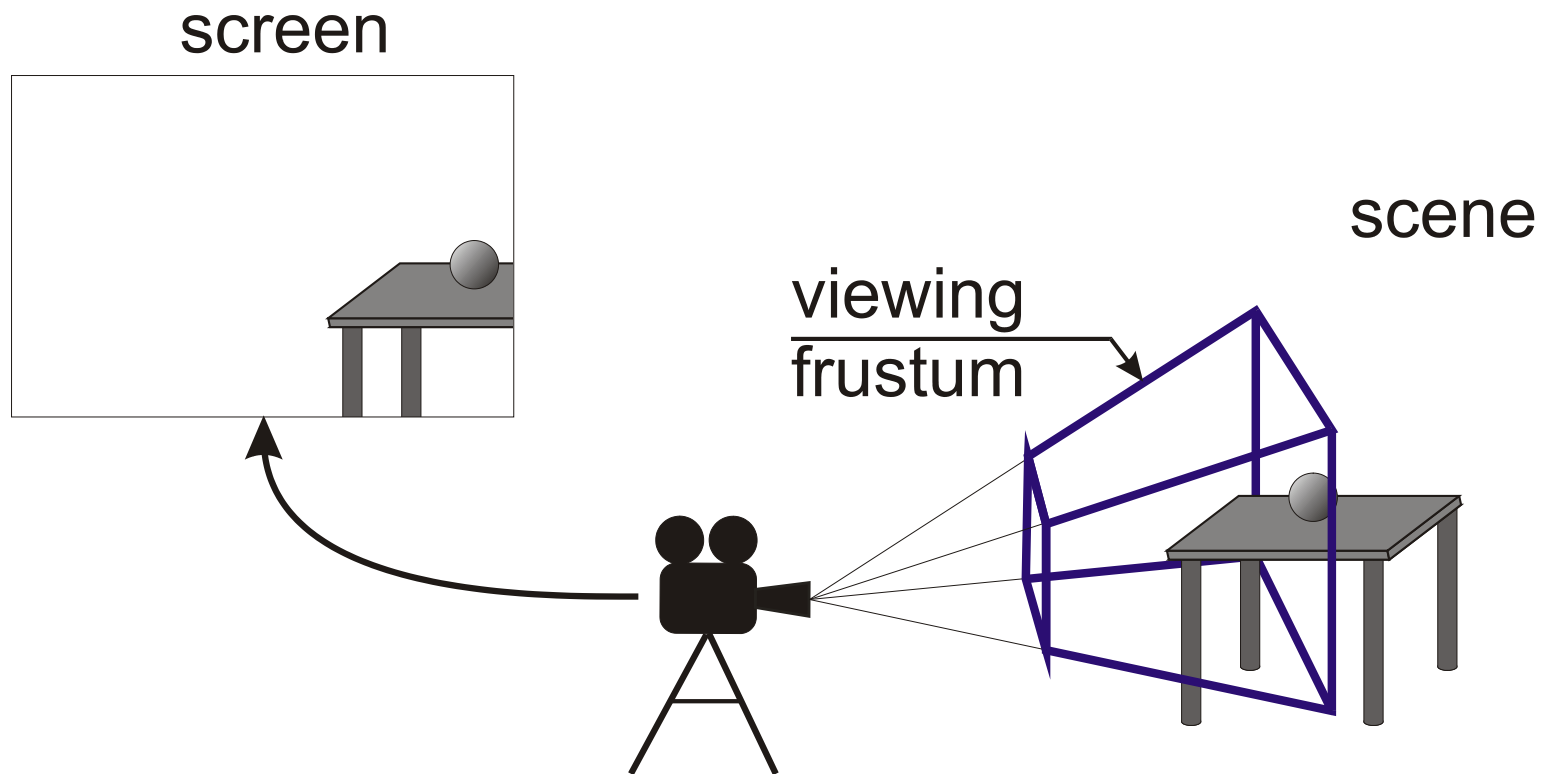
Transformace souřadnic

- Převod 3D objektů na 2D obrázky
- Potřebujeme:
 - Myslet ve 3D
 - Využít transformace (v pořadí použití):

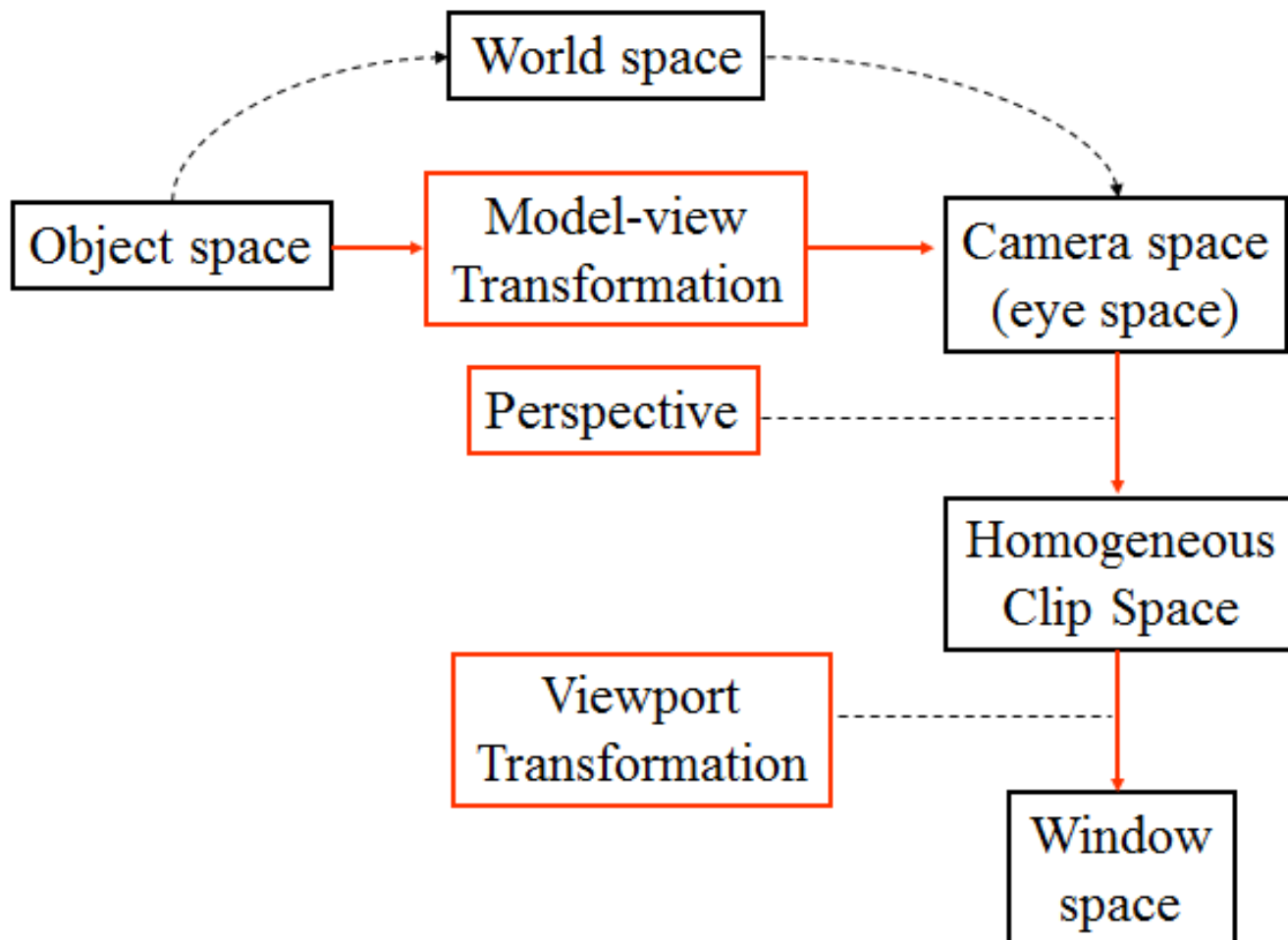


- viewing
- modeling
- projection
- clipping
- viewport

Analogie s kamerou



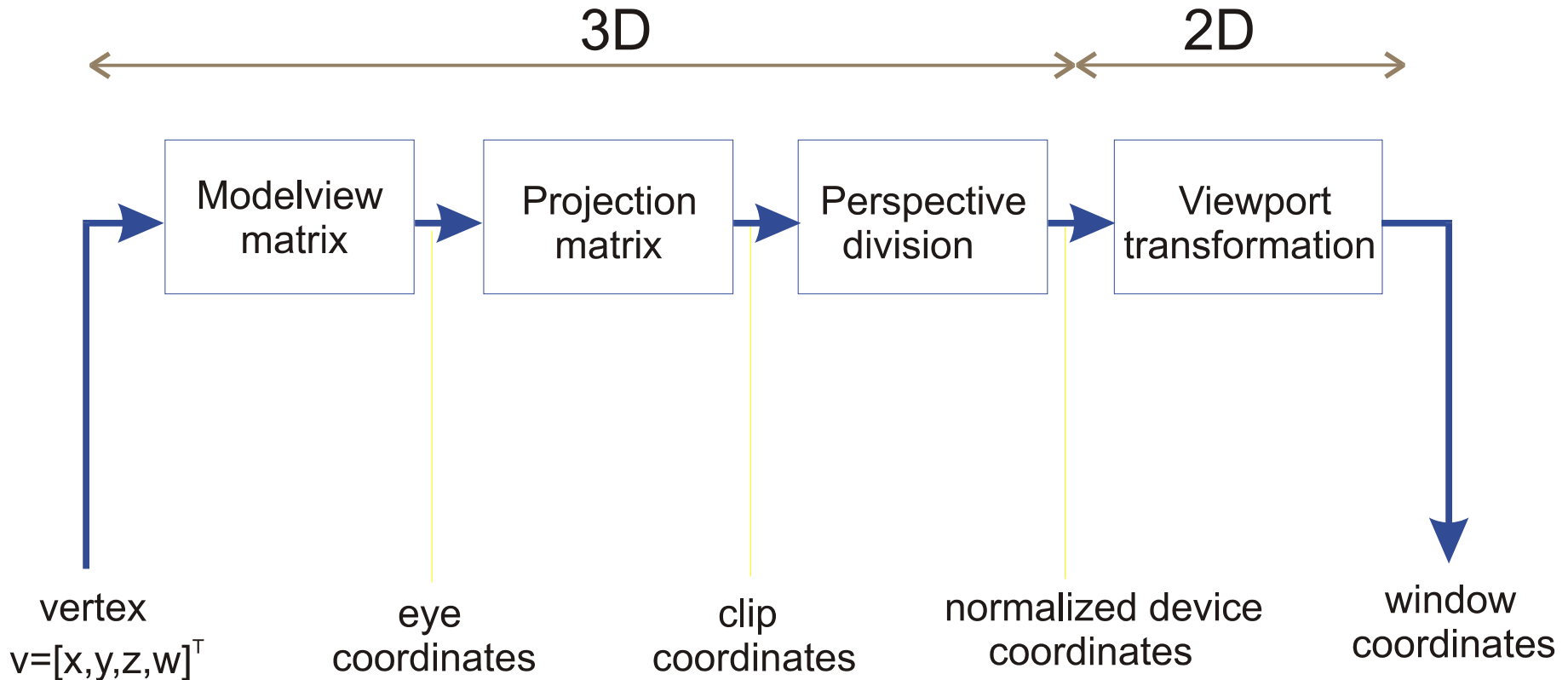
Prostory souřadnic



Prostory souřadnic

- Souřadnice objektu
- Globální souřadnice
- Souřadnice oka
- Clip (ořezávací) souřadnice
- Souřadnice okna

Transformace vrcholu



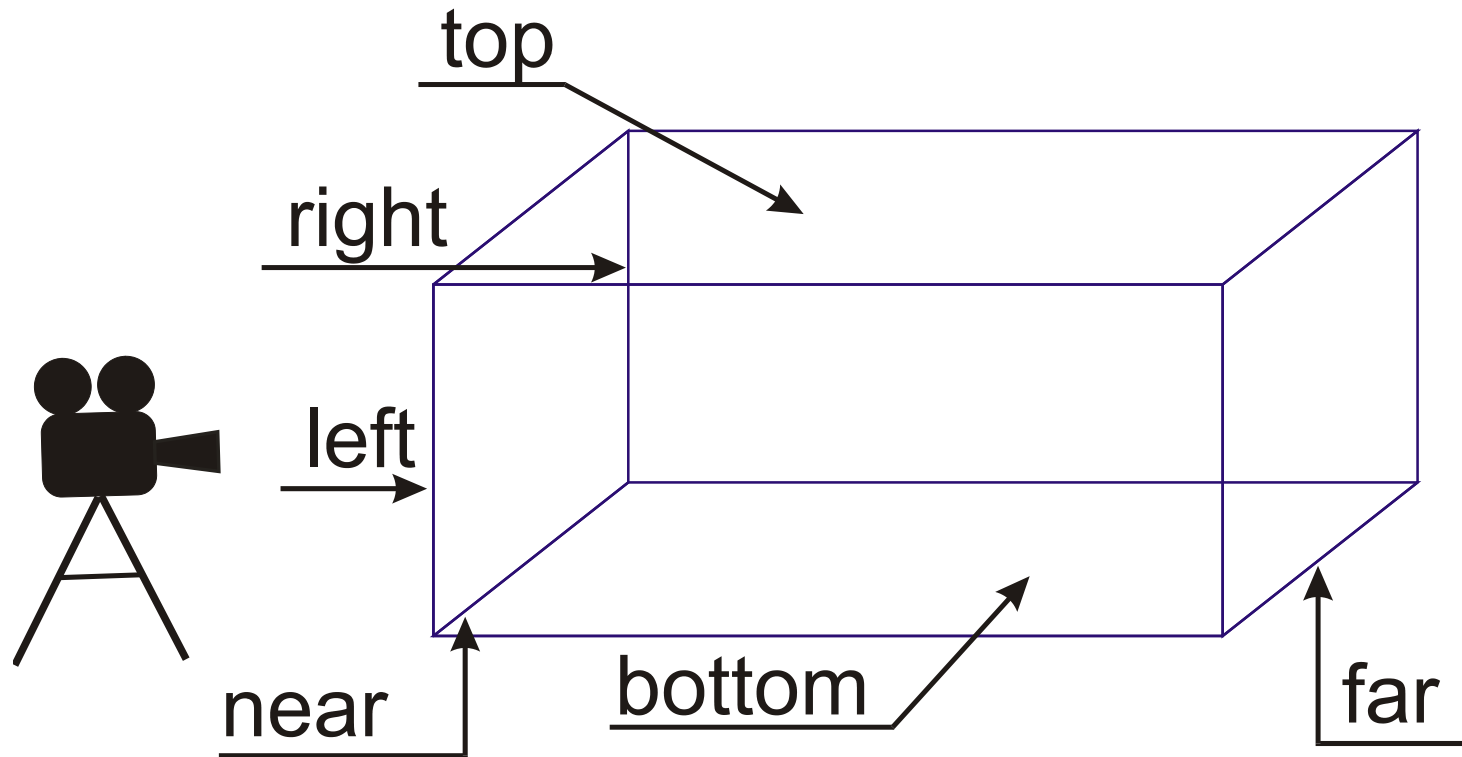
Projekce

```
glMatrixMode(GL_PROJECTION)
```

- Definice pohledového objemu, jsou vykresleny pouze objekty uvnitř tohoto objemu, ostatní ořezány
- Typy projekce:
 - Ortografická (rovnoběžná)
 - Perspektivní
 - Projekce definovaná uživatelem

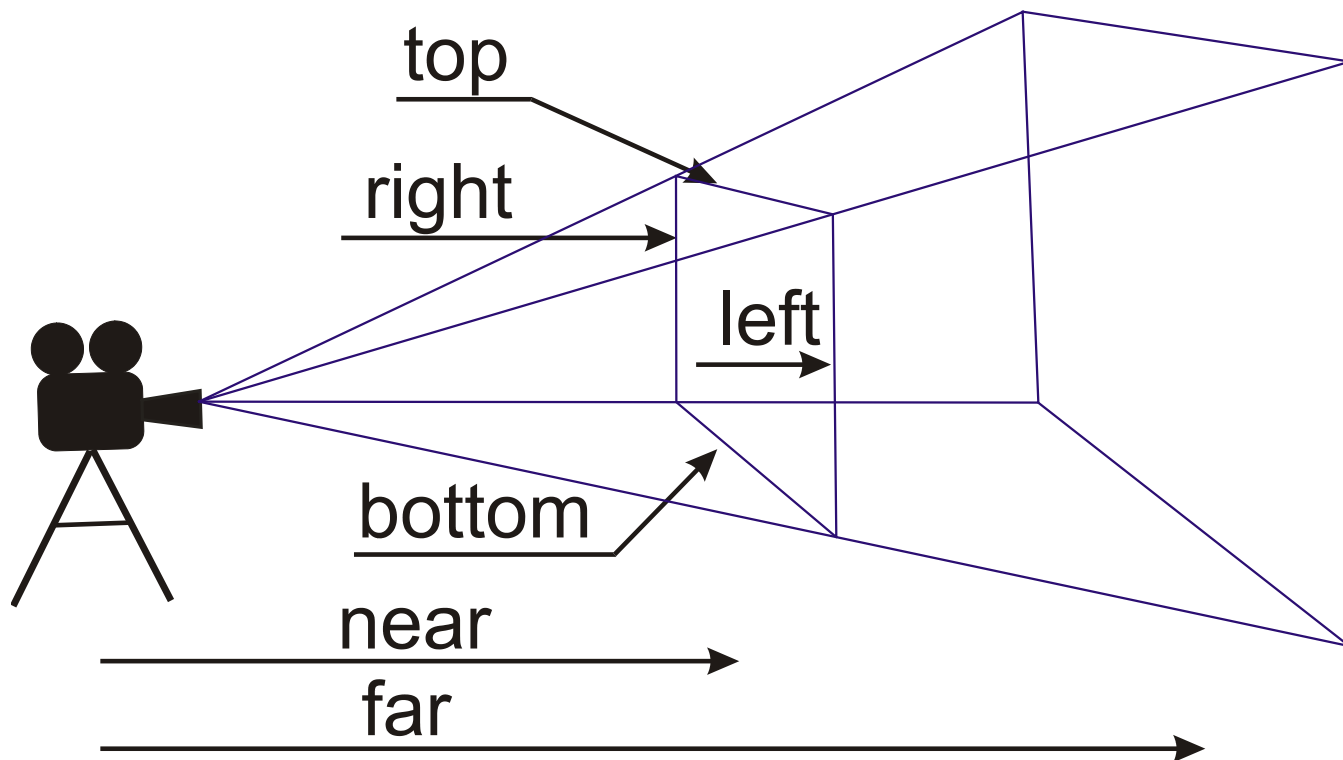
Ortografická projekce

```
void glOrtho(GLdouble left, GLdouble right,  
             GLdouble bottom, GLdouble top,  
             GLdouble near, GLdouble far)
```



Perspektivní projekce

```
void glFrustum(GLdouble left, GLdouble right,  
              GLdouble bottom, GLdouble top,  
              GLdouble near, GLdouble far)
```



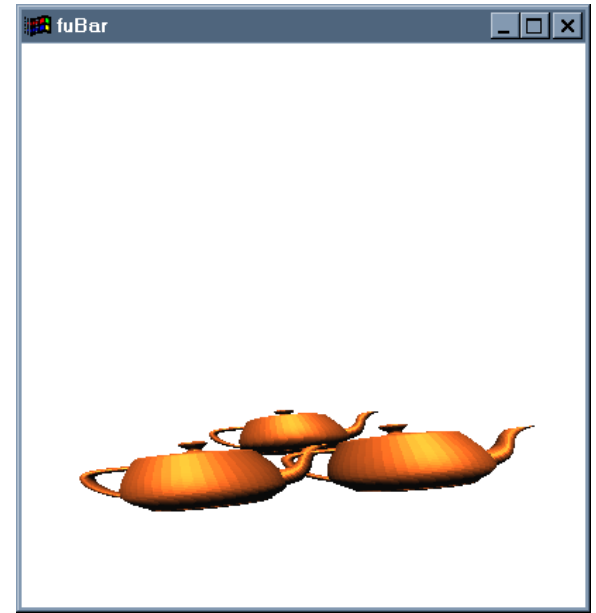
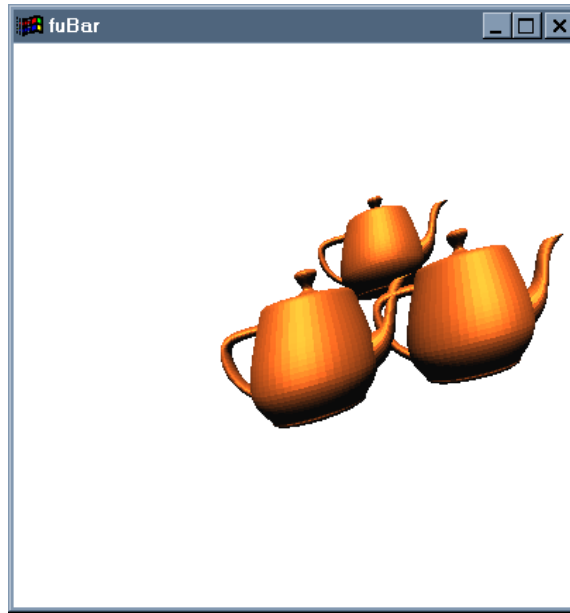
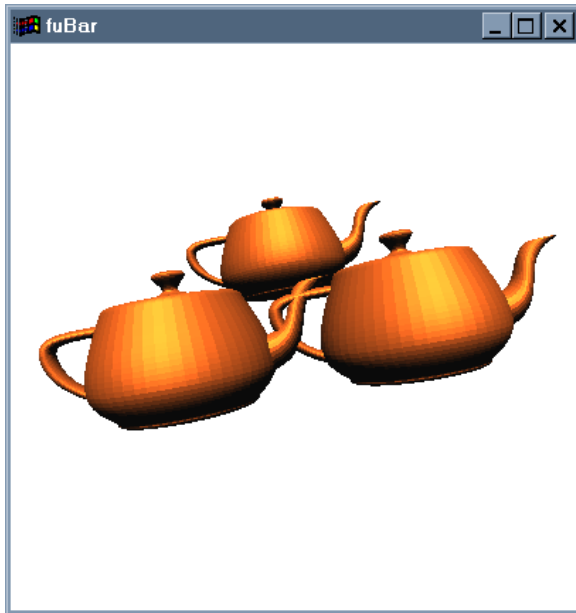
Uživatelsky definovaná projekce

```
glMatrixMode (GL_PROJECTION) ;  
glLoadMatrix (m) ;
```

- m definuje matici projekce
- Takto definujeme různé typy projekce, např. dvoubodovou perspektivu, axonometrické promítání, rybí oko apod.

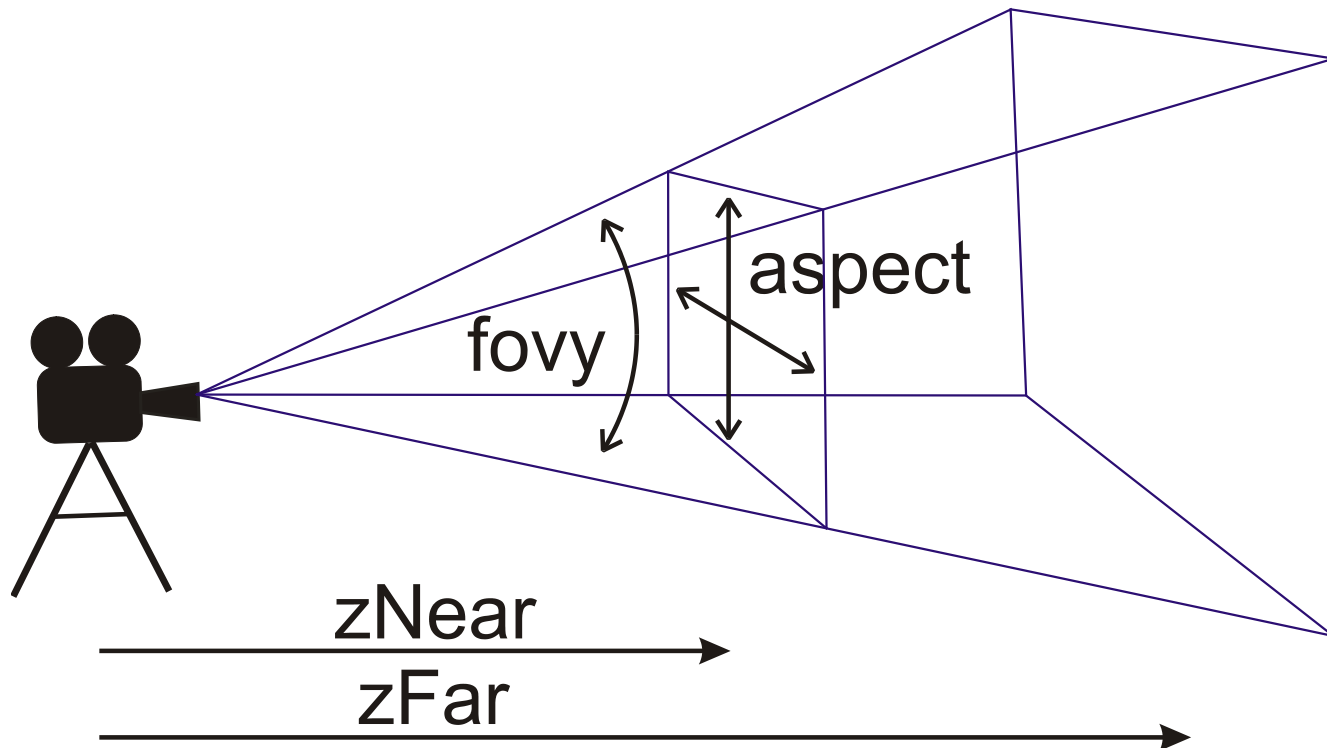
Nevýhoda glFrustum()

- glFrustum() není intuitivní díky nutnosti zadat 6 parametrů komolého jehlanu
- Možnost definovat asymetrii, může být ale matoucí



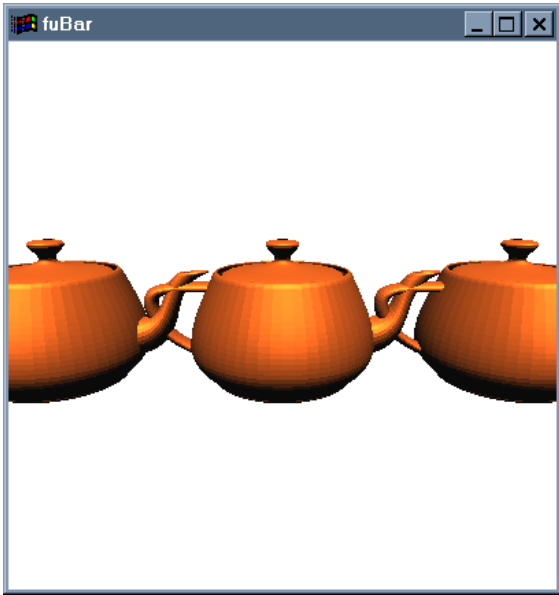
gluPerspective()

```
void gluPerspective(GLdouble fovy, GLdouble aspect,  
                   GLdouble zNear, GLdouble zFar)
```

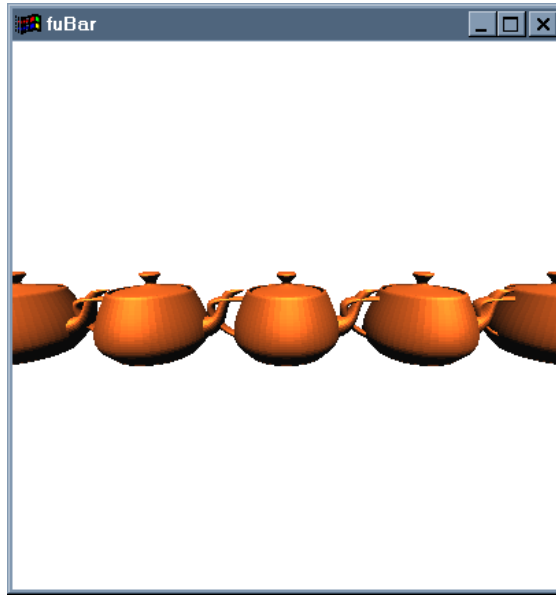


gluPerspective()

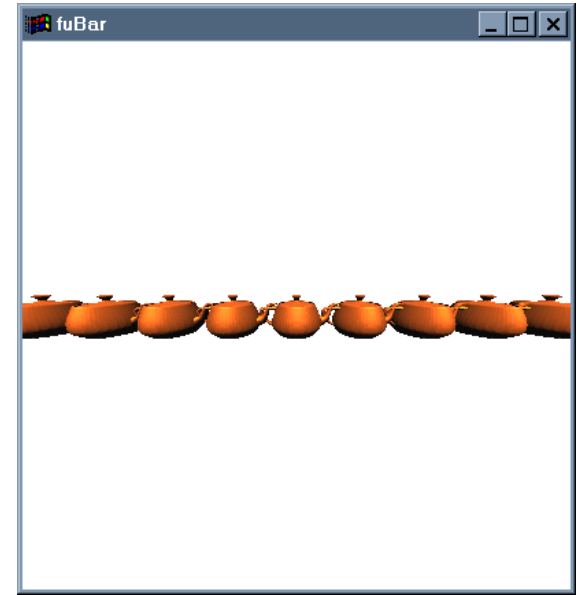
- Nelze dosáhnout asymetrie



fovy = 60



fovy = 90

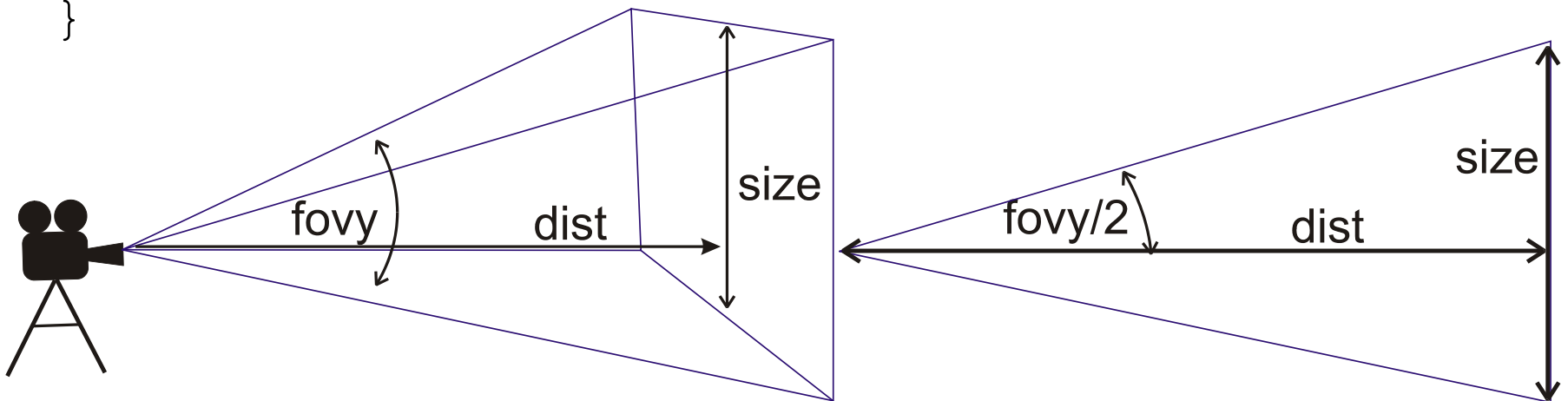


fovy = 120

Nastavení fovy

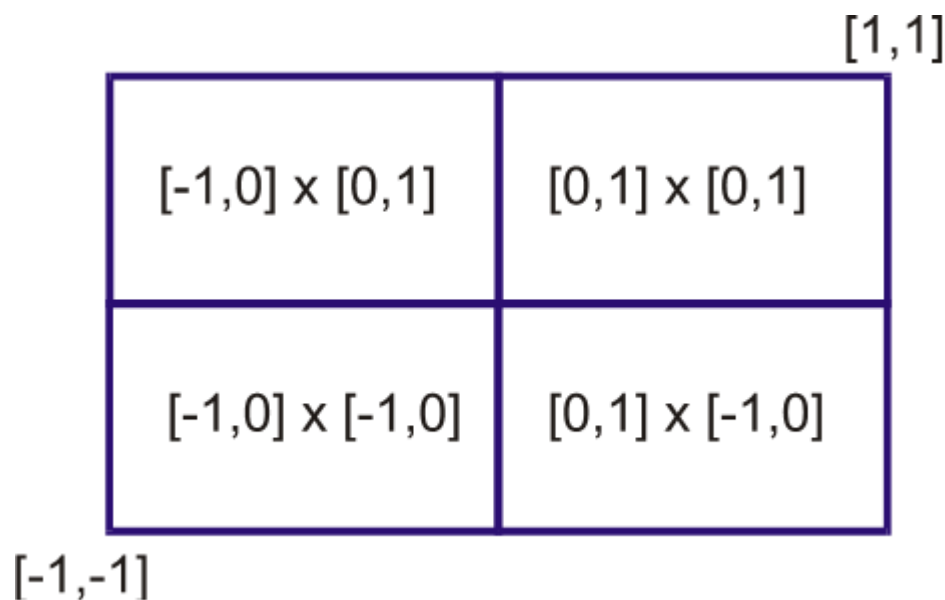
- Ze vzdálenosti a velikosti objektu

```
GLdouble FOVY(GLdouble size, GLdouble dist) {  
    GLdouble theta;  
    theta = 2.0*atan2(size/2.0, dist);  
    return (180.0*theta/π);  
}
```



Obrázek v dvojnásobném rozlišení

- Vykreslení čtyřikrát s různým nastavením projekce



Obrázek v dvojnásobném rozlišení

```
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
glFrustum(-1,0,0,1,1,10); //left upper  
Render();  
glLoadIdentity();  
glFrustum(0,1,0,1,1,10); //right upper  
Render();  
glLoadIdentity();  
glFrustum(-1,0,-1,0,1,10); //left lower  
Render();  
glLoadIdentity();  
glFrustum(0,1,-1,0,1,10); //right lower  
Render();
```


Nastavení viewportu

```
void glViewport(GLint x, GLint y,  
                GLsizei w, GLsizei h)
```

- x a y určují dolní levý roh
- w a h určují velikost obdélníku viewportu

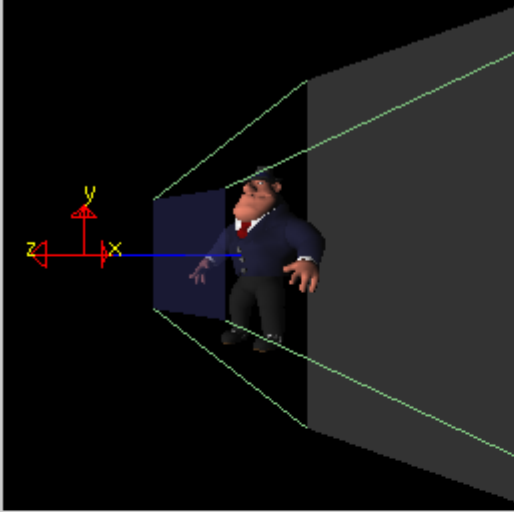
Příklad – rendering do dvou viewportů

```
int x = 600, y = 300;  
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
gluPerspective(60.0, x/(2.0*y), 1,  
10.0); // !!!  
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
glViewport(0, 0, x/2, y);  
glutSolidTeapot(0.8);  
glViewport(x/2, 0, x/2, y);  
glRotatef(60, 1, 1, 0);  
glutSolidTeapot(0.8);
```




Projekce – interaktivní ukázka

World-space view



Screen-space view



Command manipulation window

```
fovy aspect zNear zFar
gluPerspective( 60.0 , 1.00 , 1.0 , 10.0 );
gluLookAt( 0.00 , 0.00 , 2.00 ,    <- eye
           0.00 , 0.00 , 0.00 ,    <- center
           0.00 , 1.00 , 0.00 );  <- up
```

Click on the arguments and move the mouse to modify values.

Transformace souřadnice Z

```
void glDepthRange(GLclampd near, GLclampd far)
```

- *near* – mapování bližší ořezávací roviny na souřadnice okna. Implicitní hodnota 0.0.
- *far* – mapování vzdálenější přezávací roviny na souřadnice okna. Implicitní hodnota 1.0.

Zásobník matic

- Oblast vyhrazená uvnitř bloku OpenGL pro uložení 16-ti koeficientů transformační matice
- Různé zásobníky s odlišnou kapacitou pro různé typy transformačních matic

```
void glPushMatrix()
```

- Uložení aktuálně vybrané matice na vrchol zásobníku

```
void glPopMatrix()
```

- Odstranění matice z vrcholu zásobníku

Zásobník matic

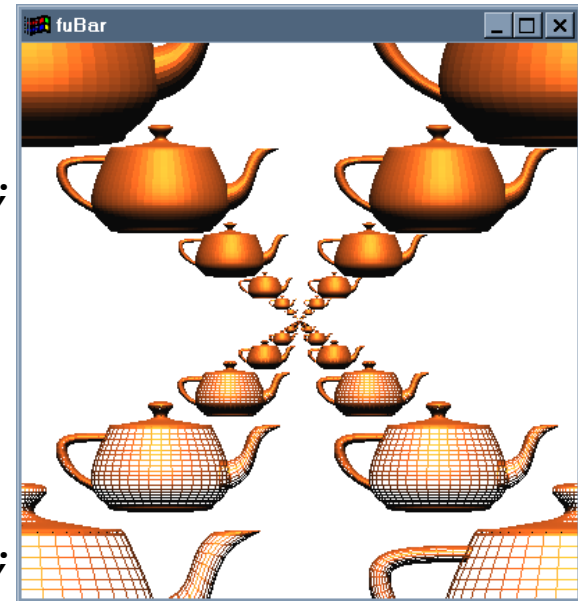
- Kapacitu jednotlivých zásobníků lze zjistit pomocí

```
glGetIntegerv(GLenum pname, GLint *params)
```

- *pname* může nabývat konstant:
 - GL_MAX_MODELVIEW_STACK_DEPTH
 - GL_MAX_PROJECTION_STACK_DEPTH
- Možné zjistit i aktuální obsazenost zásobníku:
 - GL_MODELVIEW_STACK_DEPTH
 - GL_PROJECTION_STACK_DEPTH

Příklad – zásobník matic

```
void Tea(GLfloat i){  
    glPushMatrix();  
    if (i>0.01) Tea(i/2.0);  
    glPopMatrix();  
    glPushMatrix();  
    glTranslatef(-i,-i,0);glutWireTeapot(i/2.0);  
    glPopMatrix();  
    glPushMatrix();  
    glTranslatef(-i,i,0);glutWireTeapot(i/2.0);  
    glPopMatrix();  
    glPushMatrix();  
    glTranslatef(i,-i,0);glutSolidTeapot(i/2.0);  
    glPopMatrix();  
    glPushMatrix();  
    glTranslatef(i,i,0);glutSolidTeapot(i/2.0);  
    glPopMatrix();  
}
```



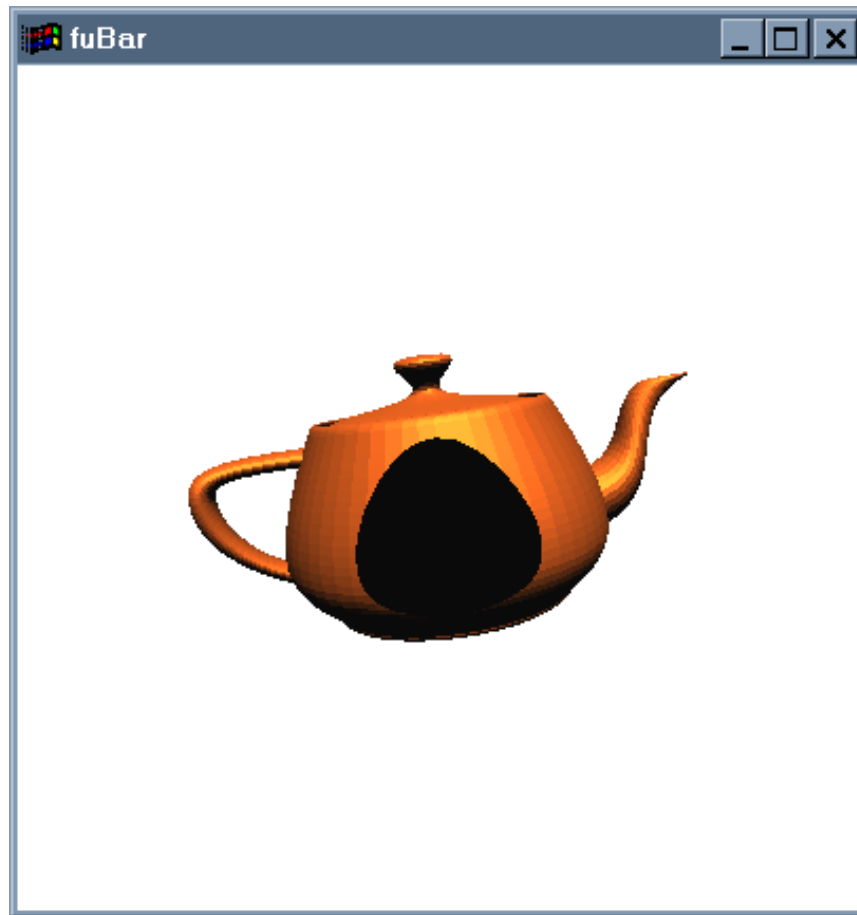
Ořezávací roviny

- Kromě šesti základních ořezávacích rovin (definovaných projekcí) můžeme definovat další ořezávací roviny:

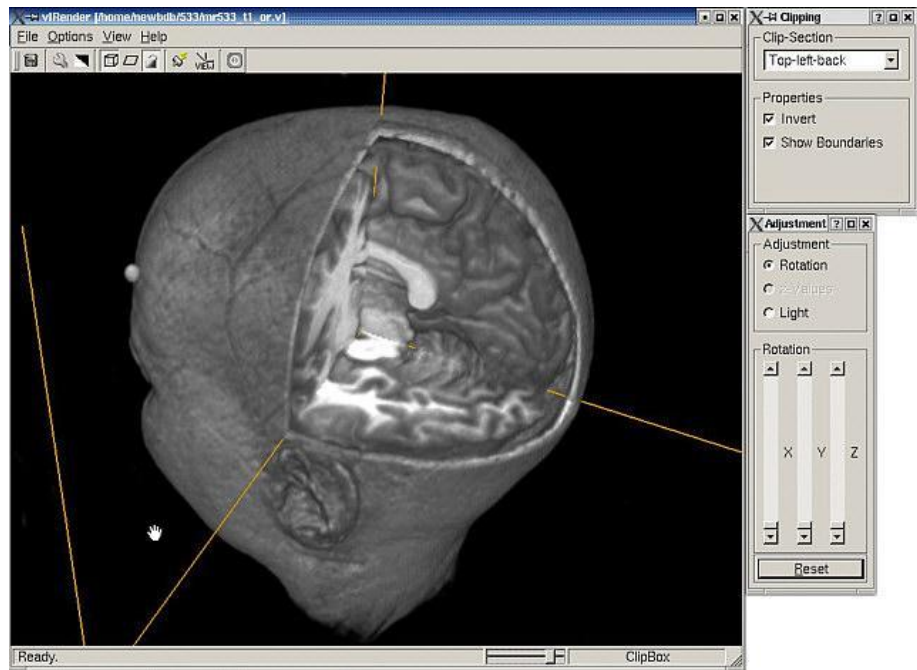
```
void glClipPlane(GLenum plane, const GLdouble *eqn)
```

- **plane** = GL_CLIP_PLANE0,..., GL_CLIP_PLANE5
- ***eqn** = ukazatel na pole čtyř parametrů A, B, C, D implicitní rovnice roviny $Ax+By+Cz+D = 0$
- Rovina musí být povolena:
 - `glEnable(GL_CLIP_PLANEi)`

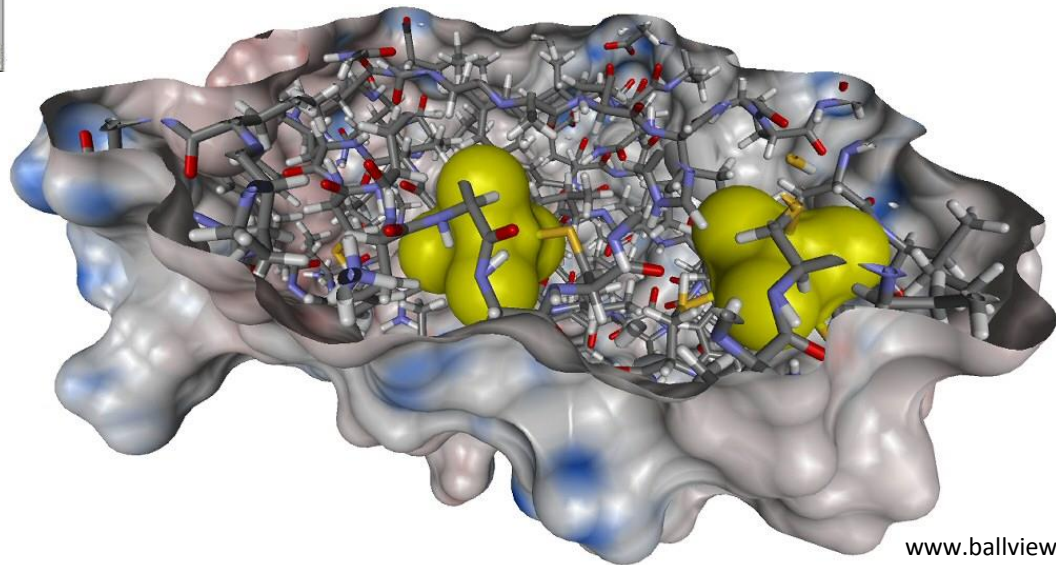
Ořezávací roviny



Ořezávací roviny



static.cbs.mpg.de



Ořezávací roviny - příklad

```
GLdouble eqn[]={-1.0,1.0,-1.0,0.0};
```

```
glLoadIdentity();
```

```
glTranslatef(0,0,-5);
```

```
glClipPlane(GL_CLIP_PLANE0,eqn);
```

```
glEnable(GL_CLIP_PLANE0);
```

```
glutSolidTeapot(1.5);
```



Stereo viewing

- Vytvoření dvou pohledů na scénu pro pravé a levé oko zvlášť
- OpenGL podporuje několik rendering bufferů
- Pro vykreslení frame ve stereomódu:
 - Displej musí tuto funkci podporovat
 - Levé/pravé oko musí být vykresleno v levém/pravém zadním bufferu
 - Zadní buffery musí být náležitě zobrazeny

Stereo viewing

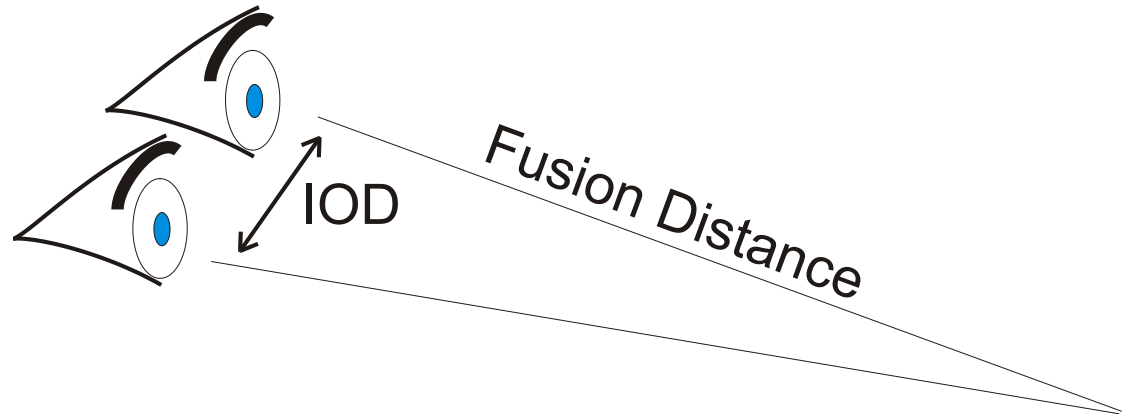
```
void glDrawBuffer(GLenum mode)
```

- *mode*:

```
GL_NONE, GL_FRONT_LEFT,  
GL_FRONT_RIGHT, GL_BACK_LEFT,  
GL_BACK_RIGHT, GL_FRONT, GL_BACK,  
GL_LEFT, GL_RIGHT,  
GL_FRONT_AND_BACK
```

Stereo viewing

- Pro určení reálného stereo pohledu musíme definovat dvě hodnoty:
 - Interocular distance (IOD)
 - Fusion distance



- Dále musíme správným způsobem umístit kameru

Příklad

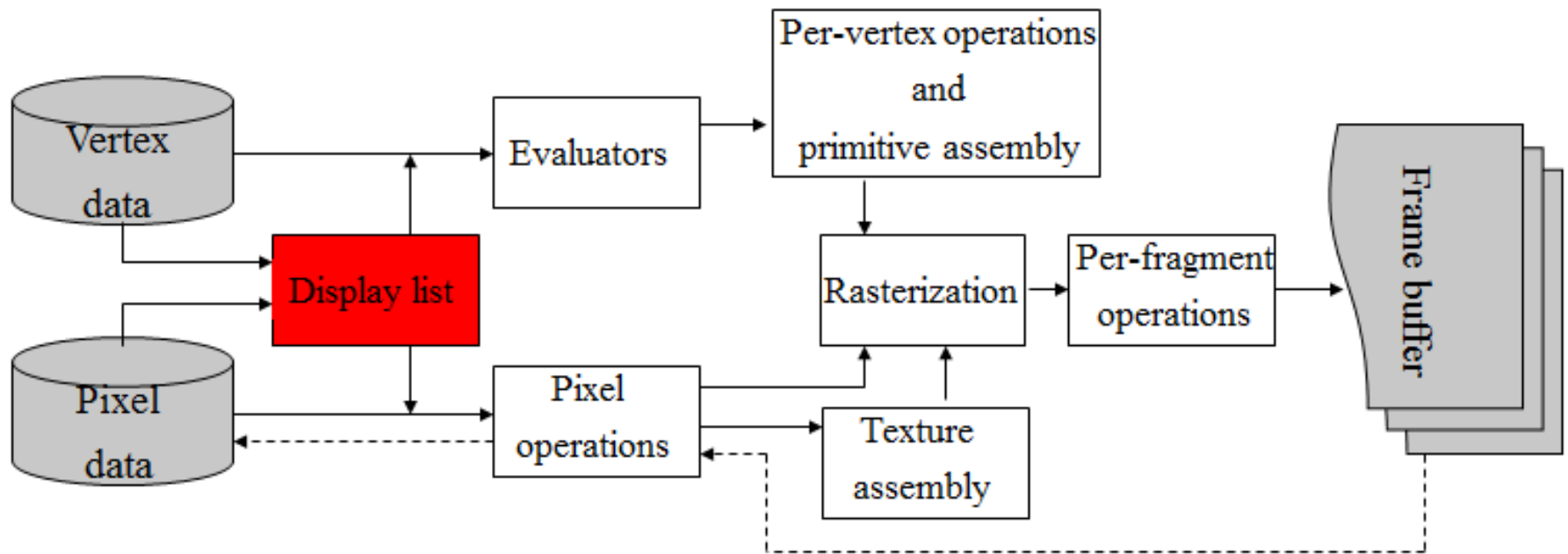
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glPushMatrix();
glDrawBuffer(GL_BACK_LEFT);
gluLookAt( -IOD/2.0, 0.0, EYE_BACK, //position
           0.0, 0.0, 0.0, //direction
           0.0, 1.0, 0.0); //up vector

RenderScene();
glPopMatrix();

glPushMatrix();
glDrawBuffer(GL_BACK_RIGHT);
gluLookAt( IOD/2.0, 0.0, EYE_BACK, //position
           0.0, 0.0, 0.0, //direction
           0.0, 1.0, 0.0); //up vector

RenderScene();
glPopMatrix();
```

Display listy

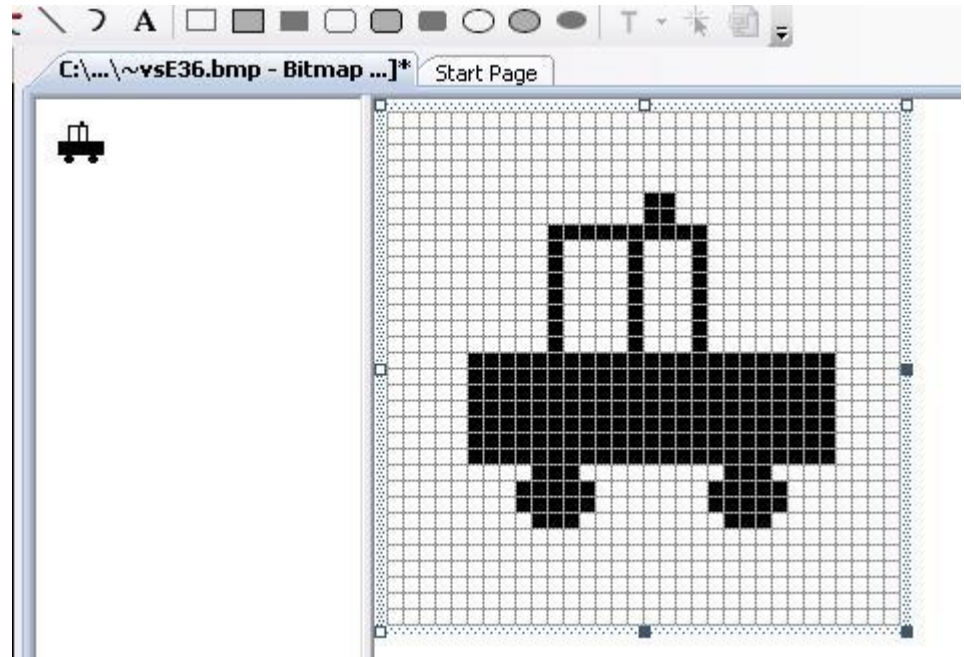


Display listy

- Skupina příkazů uložených pro pozdější provedení
- Odložený mód (retained mode)
- Efektivní v síťovém prostředí
- Odložený mód je vždy alespoň tak rychlý jako okamžitý mód (immediate mode)
- Vytvořený display list již nemůže být modifikován
- Display listy mohou být vytvářeny hierarchicky

Display listy

- Operace, které jsou za použití display listů efektivnější:
 - Maticové operace
 - Bitmapy a textury
 - Světla, materiály
 - Polygon stippling



Display listy

- Vytvoření:

```
void glNewList(GLuint list, GLenum mode)
```

- *list* = identifikační číslo listu, zadané uživatelem
- *mode* = má-li se display list pouze vytvořit (GL_COMPILE) nebo vytvořit a hned provést (GL_COMPILE_AND_EXECUTE)

- Ukončení:

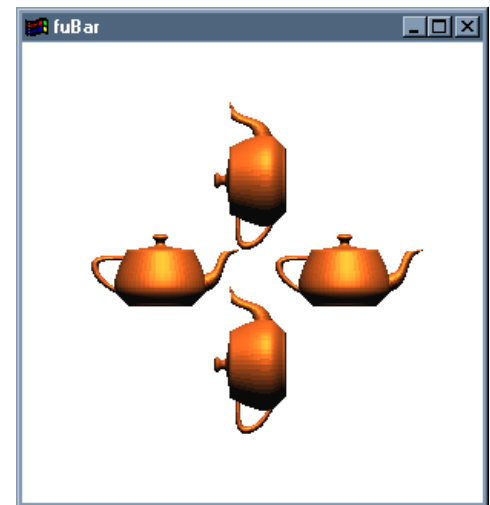
```
void glEndList()
```

- Vyvolání:

```
void glCallList(GLuint list)
```

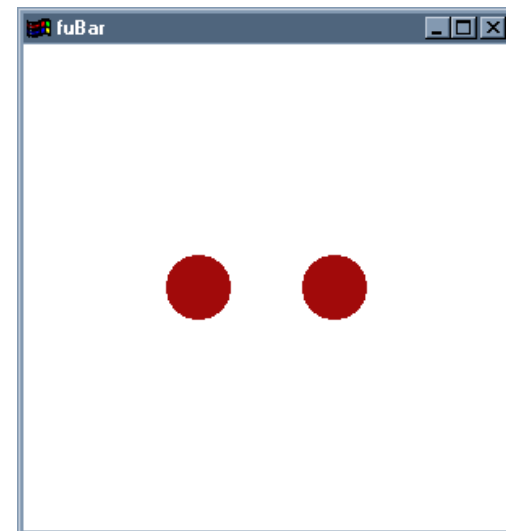
Display listy - příklad

```
glNewList(1, GL_COMPILE) ; //create new  
                             list  
  
    glTranslatef(-1, 0, 0) ;  
    glutSolidTeapot(0.5) ;  
    glTranslatef(2, 0, 0) ;  
    glutSolidTeapot(0.5) ;  
    glTranslatef(-1, 0, 0) ;  
glEndList() ; //end display list  
glCallList(1) ; //horizontal  
glRotatef(90, 0, 0, 1) ;  
glCallList(1) ; //vertical
```



Display listy - příklad

```
glNewList(1, GL_COMPILE);  
    glColor3f(1.0, 0.0, 0.0);  
    glBegin(GL_TRIANGLE_FAN);  
    glVertex3f(0,0,0);  
    for (i=0; i<=MAX; i++) //very slow cycle !!  
        glVertex3f(sin(i*2π/MAX)/3.0, cos(i*2π/MAX)/3.0, 0);  
    glEnd();  
glEndList(); // end display list  
glTranslatef(-0.7, 0, 0); //very efficient piece of code  
glCallList(1); //left  
glTranslatef(1.4, 0, 0);  
glCallList(1); //right  
glFlush();
```



Display listy

- V display listech se nemohou používat následující příkazy:
 - **Dotazové příkazy** (glIs*(), glGet*(), glReadPixels(), glReadBuffer())
 - **Manipulace s display listy** (glDeleteLists(), glIsList(), glGenLists())
 - **a další...** (glFlush(), glFinish(), glRenderMode(), glPixelStore(), glFeedbackBuffer())

Display listy

- Při vytváření nového display listu nesmíme použít index, který je již obsazený – původní display list s tímto indexem bychom přepsali

```
GLboolean glIsList(GLuint list)
```

- TRUE, pokud je index *list* obsazen

```
GLuint glGenList(GLsizei range)
```

- Alokuje spojitý rozsah indexů, vrácené číslo je první index

Display listy

- Smazání display listů v daném rozsahu

```
void glDeleteLists(GLuint list, GLsizei range)
```

- *list* = počáteční index display listů pro smazání
 - Tyto indexy jsou opět volné pro další použití
- Display listy mohou být **hierarchické** = jeden display list může obsahovat volání jiných zkompilovaných display listů

```
glNewList(7, GL_COMPILE);  
    for (i=0; i<7; i++) glCallList(i);  
glEndList();
```


Display listy

- Vyhnutí se nekonečné rekurzi – definice limitu vnoření

```
glGetIntegerv(GL_MAX_LIST_NESTING, GLint *dta)
```

- **Vícenásobné** spouštění display listů

```
void glListBase(GLuint base)
```

- *base* = offset připočítávaný k indexům display listu v `glCallLists()`

```
void glCallLists(GLsizei n, GLenum type, const GLvoid *lists)
```

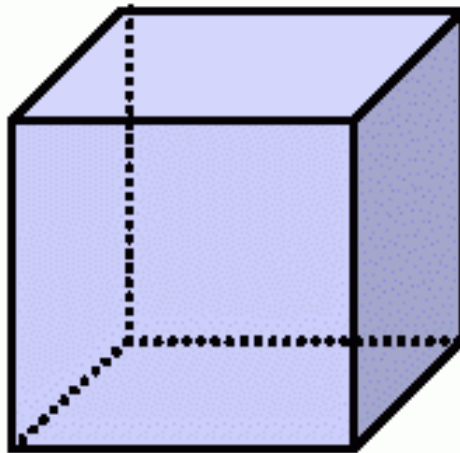
- *n* = počet spuštěných display listů, *lists* = pole indexů, *type* = typ prvků v *lists*

Vertex Arrays (pole vrcholů)

- Složitější objekty nutno složit z 10ti základních grafických primitiv, každému vrcholu přiřazujeme vlastnosti:
 - Pozice vrcholu v rovině či prostoru (příkaz `glVertex*()`)
 - Barva vrcholu (příkaz `glColor*()`)
 - Index barvy vrcholu v barevné paletě (`glIndex*()`)
 - Normálový vektor ve vrcholu (`glNormal*()`)
 - Souřadnice do textury (`glTexCoord*()`)
 - Mapovací souřadnice (`glEvalCoord*()`)
 - 1D nebo 2D bod v síti (`glEvalPoint*()`)
 - Optické vlastnosti materiálu (`glMaterial()`)
 - Vlastnosti hrany – viditelnost (`glEdgeFlag()`)

Vertex Arrays

- Nevýhoda – volání několik tisíc i desítek tisíc těchto funkcí (podle složitosti scény)
 - např. pro krychli voláme 24krát funkci `glVertex*()`



šest stran a osm vrcholů,
24 volání funkce `glVertex()` !

Vertex Arrays

- Možnosti redukce počtu volání funkcí:
 - Nastavení stavu celému bloku místo jednotlivým příkazům
 - Využití grafických primitiv redukujících počet vrcholů
 - Display listy
 - **Pole vrcholů** (vertex arrays)

Vertex Arrays

- Uložení vrcholů do vhodně organizovaného pole
- Poslání dat vrcholů pomocí několika málo funkčních volání
- Možnost měnit data vrcholů mezi jednotlivými snímky
- Minimalizace počtu volaných funkcí a dat procházejících přes pipeline
- Specifikace množství dat v několika málo polích

Vertex Arrays

1. Aktivace (až šesti) polí pro různé typy dat.
2. Vložení dat do polí.
3. Vykreslení geometrie dat. **V client-server modelu jsou data přemístěna do adresového prostoru serveru.** Vykreslování probíhá jedním ze tří způsobů:
 - i. Samostatný přístup k jednotlivým prvkům polí
 - ii. Vytvoření seznamu jednotlivých prvků polí
 - iii. Sekvenční zpracování prvků polí

Vertex Arrays - aktivace

- Povolení používání

```
void glEnableClientState(GLenum array)
```

– *array* = jedna z konstant GL_VERTEX_ARRAY,
GL_COLOR_ARRAY, GL_INDEX_ARRAY,
GL_NORMAL_ARRAY, GL_TEXTURE_COORD_ARRAY,
GL_EDGE_FLAG_ARRAY

- Zakázání užívání

```
void glDisableClientState(GLenum array)
```

- V praxi maximálně 4 pole současně

Vertex Arrays - aktivace

- Proč nebyly použity funkce `glEnable()` a `glDisable()` ?
 - `glEnable()` a `glDisable()` mohou být uloženy v Display listu, zatímco specifikace Vertex Arrays nemůže, protože data zůstávají na straně klienta

Vertex Arrays – vkládání dat do polí

- Druhým krokem je vlastní vytvoření pole vrcholů
- Vkládat hodnoty do polí lze dvěma způsoby:
 - Šesti různými funkcemi `gl*Pointer()`, z nichž každá specifikuje přístup do jednoho pole v prostoru klienta
 - Pomocí jedné komplexní funkce pro prokládaná pole – `glInterleavedArrays()`

Vkládání dat do polí – 6 funkcí

- Podle typu uložených dat existují funkce:

`glVertexPointer()`

`glColorPointer()`

`glIndexPointer()`

`glNormalPointer()`

`glTexCoordPointer()`

`glEdgeFlagPointer()`

Vkládání dat do polí – glVertexPointer

```
void glVertexPointer(GLint size, GLenum type,  
                    GLsizei stride, const GLvoid *pointer);
```

- *size* = počet souřadnic na jeden vrchol
- *type* = datový typ položek
- *stride* = mezera v bytech mezi jednotlivými položkami
- *pointer* = ukazatel na pole s informacemi o vrcholech

Příklad

```
static GLint vertices[] = {100, 100, 100,  
                           100, 50, 200, ... }  
  
glEnableClientState(GL_VERTEX_ARRAY);  
glVertexPointer(3, GL_INT, 0, vertices);
```

- Funkce `glVertexPointer` určí, kde se nachází prostorová data a jak vypadají. Není určena délka pole.

Ostatní funkce

- `void glColorPointer(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);`
- `void glIndexPointer(GLenum type, GLsizei stride, const GLvoid *pointer);`
- `void glNormalPointer (GLenum type, GLsizei stride, const GLvoid *pointer);`
- `void glTexCoordPointer(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);`
- `void glEdgeFlagPointer(GLsizei stride, const GLvoid *pointer);`

Ostatní funkce

Příkaz:	Velikost:	Typ: (GL_*)
gl Vertex *	2, 3, 4	_SHORT, _INT, _FLOAT _DOUBLE
gl Normal *	3	_BYTE, _SHORT, _INT, _FLOAT, _DOUBLE
gl Color *	3, 4	(_UNSIGNED) _BYTE, () _SHORT, () _INT, _FLOAT, _DOUBLE
gl Index *	1	() _BYTE, _SHORT, _INT, _FLOAT, _DOUBLE
gl TexCoord *	1, 2, 3, 4	_SHORT, _INT, _FLOAT _DOUBLE
gl EdgeFlag *	1	musí být typu GLboolean

Vkládání dat do polí – prokládaná pole

```
void glInterleavedArrays(GLenum format,  
                        GLsizei stride, void *pointer);
```

- *format* je jedna ze 14 předdefinovaných konstant:
GL_V2F, GL_V3F, GL_C4UB_V2F, GL_C4UB_V3F,
GL_C3F_V3F, GL_N3F_V3F, GL_C4F_N3F_V3F, GL_T2F_V3F,
GL_T4F_V4F, GL_T2F_C4UB_V3F, GL_T2F_C3F_V3F,
GL_T2F_N3F_V3F, GL_T2F_C4F_N3F_V3F, GL_T4F_C4F_N3F_V4F
- *stride* je offset v bytech mezi následujícími vrcholy. Je-li 0, pak se předpokládá, že vrcholy jsou těsně za sebou.
- *pointer* je adresa v paměti obsahující první souřadnici.

Vkládání dat do polí – prokládaná pole

Příkaz `glInterleavedArrays()` provede současně aktivaci (pomocí `glEnableClientState()` a `glDisableClientState()`) všech použitých polí a přiřadí jim data v paměti.

Příklad:

```
static GLfloat intertwined[]=  
    {1.0, 0.5, 0.7, 100.0, 100.0, 100.0,  
     0.4, 0.6, 1.0, 50.0, 20.0, 100.0, ... }  
...  
glInterleavedArrays(GL_C3F_V3F, 0,  
&intertwined[0]);
```


Vykreslení geometrie dat

- Dokud nedojde k dereferenci polí, pole zůstávají na straně klienta.
- V kroku 3 je zjištěn obsah polí, poslán serveru a pak předán grafické pipeline.
- Prvky pole lze vybrat třemi různými způsoby:
 - `glArrayElement(GLint i)`
 - `glDrawElements(GLenum mode, GLsizei count, GLenum type, void *indices)`
 - `glDrawArrays(GLenum mode, GLint first, GLsizei count)`

Vykreslení geometrie dat

```
void glVertexElement(GLint ith);
```

- Příkaz sloužící k dereferenci jednoho prvku
- *ith* = pořadové číslo vrcholu, který bude zpracován pro všechna aktuálně používaná pole
- Stejný efekt jako volání funkce glVertex(), glColor(), ...

Vykreslení geometrie dat

```
glEnableClientState(GL_VERTEX_ARRAY);  
glEnableClientState(GL_COLOR_ARRAY);  
  
glColorPointer(3, GL_FLOAT, 0, colors);  
glVertexPointer(3, GL_FLOAT, 0, vertices);  
  
glBegin(GL_TRIANGLES);  
glArrayElement(1); // = glVertex3f(vertices+1*3);  
glColor3f(...);  
glArrayElement(5);  
glArrayElement(3);  
glEnd();
```

POZOR! Pokud dojde ke změně obsahu polí mezi `glBegin()`, `glEnd()`, pak není jisté, která data obdržíme.

Vykreslení geometrie dat

```
void glDrawElements(GLenum mode, GLsizei count,  
                    GLenum type, void *indices);
```

- *mode* určuje druh vytvářených primitiv (GL_POLYGON, ...)
- *count* je počet prvků pro které se bude provádět dereference
- *type* je buď GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT nebo GL_UNSIGNED_INT určuje datový typ indexového pole
- *indices* indexové pole

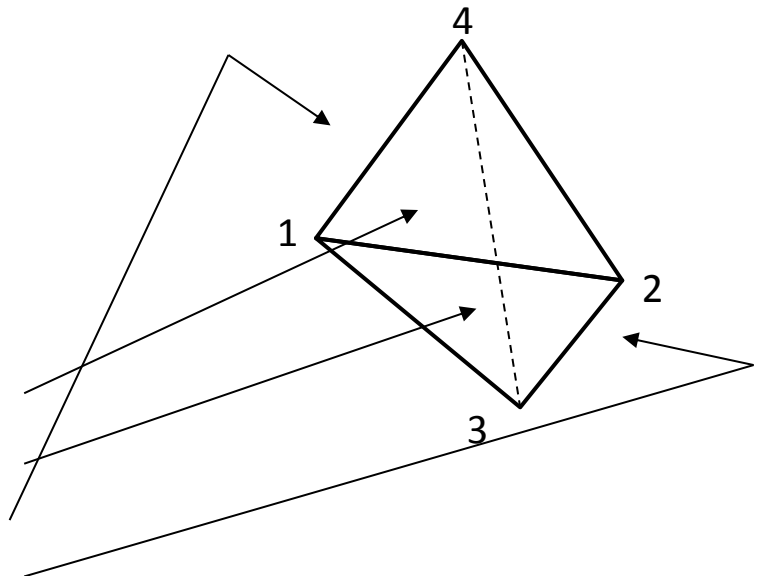
Vykreslení geometrie dat

Efekt `glDrawElements()` je ekvivalentní k:

```
int i;  
glBegin(mode);  
for (i= 0; i< count; i++)  
    glVertexElement(indices[i]);  
glEnd();
```

Příklad:

```
static GLubyte firstside[] = {1,2,4};  
static GLubyte secondside[] = {1,2,3};  
static GLubyte thirdside[] = {1,3,4};  
static GLubyte fourthside[] = {2,3,4};  
  
glDrawElements(GL_TRIANGLES, 3, GL_UNSIGNED_BYTE, firstside);  
glDrawElements(GL_TRIANGLES, 3, GL_UNSIGNED_BYTE, secondside);  
glDrawElements(GL_TRIANGLES, 3, GL_UNSIGNED_BYTE, thirdside);  
glDrawElements(GL_TRIANGLES, 3, GL_UNSIGNED_BYTE, fourthside);  
static GLubyte allindices[] = {1,2,4, 1,2,3, 1,3,4, 2,3,4};  
glDrawElements(GL_TRIANGLES, 12, GL_UNSIGNED_BYTE, allindices);
```



Vykreslení geometrie dat

```
void glDrawRangeElements(GLenum mode, GLuint start,  
                        GLuint end, GLsizei count, GLenum type,  
                        void *indices);
```

- *mode*, *count*, *type* a *indices* mají stejný význam jako v `glDrawElements()`
- *[start - end]* rozsah akceptovatelných hodnot pro *indices*

Vykreslení geometrie dat

Dereference omezeného seznamu prvků polí:

- Je chybou, pokud se prvky z indices odkazují mimo daný rozsah. OpenGL implementace však nemusí tuto chybu rozpoznat a hlásit.
- Implementace je omezena = vyšší efektivita
- Pomocí `glGetIntegerv(GL_MAX_ELEMENTS_VERTICES)` můžeme zjistit doporučené maximum počtu vrcholů a pomocí `glGetIntegerv(GL_MAX_ELEMENTS_INDICES)` doporučené maximum počtu indexů v poli.
- Jestliže je *start–end+1* větší než doporučené maximum vrcholů, nebo *count* větší než doporučené maximum indexů, bude se pořád vykreslovat korektně, ale se ztrátou efektivity.

Vykreslení geometrie dat

```
void glDrawArrays(GLenum mode, GLint first, GLsizei count);
```

- Dereference sekvenčního pole prvků
- Vytvoří sekvenci geometrických primitiv s použitím prvků polí počínaje *first* a konče *first + count - 1*.
- Efekt `glDrawArrays()` je ekvivalentní k:

```
int i;  
glBegin(mode);  
for (i= 0; i< count; i++)  
    glArrayElement(first + i);  
glEnd();
```


Vykreslení geometrie dat

```
void glMultiDrawArrays(GLenum mode, GLint *first,  
                       GLsizei *count, GLsizei primcount);
```

- Dereference od verze OpenGL 1.4
- *first* je ukazatel na pole počátečních indexů
- *count* je ukazatel na pole počtu vykreslovaných primitiv
- *primcount* Určuje velikost polí *first* a *count*