17) Správa menu v GLUTu. Co všechno obsahuje a jak se používá správa menu v GLUTu.

Glut podporuje základní kaskádové POP-UP menu. Položky menu není možné upravovat (přidávat, mazat a měnit) v době, kdy je okno používáno. Menu může obsahovat běžné položky a podmenu.

```
int PodMenu1;
PodMenu1 = glutCreateMenu(OnClickFunc);  //Vytvoří nové menu (Obslužná_funkce)
glutAddMenuEntry("PodMenuPolozka1", 1);  //Přidá položku (Název, Hodnota)
glutAddMenuEntry("PodMenuPolozka2", 2);
glutCreateMenu(OnClickFunc);
glutAddSubMenu("PodMenu", PodMenu1);     //Přidá položku odkazující se na podmenu
glutAddMenuEntry("Polozka1",3);
glutAttachMenu(GLUT_RIGHT_BUTTON);       //Tlačítko myši, při kterém se menu vyvolá
                                         //GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON

void OnClickFunc(int val)
{ switch (val) {
    case 1: printf(„Vybrána položka 1“); break;
  }
}

void glutDetachMenu(int button)      //Odpojí aktivační tlačítko myši
void glutDestroyMenu(int menu_id)    //Zruší menu
void glutSetMenu(int menu_id)        //Nastaví id pro menu
int glutGetMenu(void)                //Zjistí id aktuálního menu
```

30) Popište „Vertex Arrays“, princip a základní kroky použití.

Vertex Arrays (pole vrcholů) je skupina nástrojů (funkcí) určených k efektivnější implementaci vkládání dat. Snaží se minimalizovat počet volaných funkcí a dat procházejících přes pipeline. Efektivita vzrůstá u síťových aplikací. Vertex Arrays funkce umožňují specifikovat mnoho dat (vztažených k vrcholům) v několika málo polích.

Základní kroky:
1. Aktivace (až šesti) polí pro různé typy dat.
2. Vložení dat do polí. (GL_[VERTEX/COLOR/INDEX/NORMAL/TEXTURE_COORD/EDGE_FLAG]_ARRAY)
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.** Seekvenční zpracování prvků polí.

31) Popište vkládání dat do „Vertex Arrays“, uveďte příkazy. Nápověda: ??(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer), ??(GLenum format, GLsizei stride, void *pointer)

Lze provést dvěma způsoby:
a) 6-ti různými funkcemi gl*Pointer(…) z nichž každá specifikuje přístup do jednoho pole v prostoru klienta.
b) Pomocí jedné komplexní funkce pro prokládaná pole (InterleavedArrays) glInterleavedArrays(…)

ad a)
Nejprve aktivujeme odpovídající pole pomocí **glEnableClientState**(GL_***_ARRAY);
Na konci plnění dat pole deaktivujeme (**glDisableClientState**).

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

size – počet prvků, například počet souřadnic na jeden vrchol
type - specifikuje datový typ, například GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE...
pointer - adresa v paměti obsahující první souřadnici.
stride - offset v bytech mezi následujícími vrcholy. Je-li 0, pak se předpokládá, že vrcholy jsou těsně za sebou. Offset se počítá od ukazatele na první souřadnici!

Ad b)
Prokládaná pole (Interleaved Arrays). Příkaz **glInterleavedArrays**() provede současně aktivaci (pomocí **glEnableClientState**() a **glDisableClientState**()) všech použitých polí a přiřadí jim data v paměti.

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

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.
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

32) Popište vykreslování dat z „Vertex Arrays“, uveďte příklady. Nápověda: ??(GLint ith) ??(GLenum mode, GLsizei count, GLenum type, void *indices) ??(GLenum mode, GLuint start, GLuint end, GLsizei count, GLenum type, void *indices), ??(GLenum mode, GLint first, GLsizei count)

void **glArrayElement**(GLint **ith**);
- slouží k dereferenci (vykreslení) jednoho prvku z aktivních polí.
ith – pořadové číslo vrcholu, který bude zpracován ze všech aktuálně používaných polí.

void **glDrawElements**(GLenum **mode**, GLsizei **count**, GLenum **type**, void ***indices**);
- slouží k dereferenci prvků z aktivních polí jejichž pořadové čísla jsou uložený v indexovém poli.
- každý prvek daný indexovým polem je vykreslen pomocí **glArrayElement**.

mode - určuje druh vytvářených primitiv (GL_POLYGON, …)
count - počet prvků pro které se bude provádět dereference
type – GL_[UNSIGNED_BYTE/UNSIGNED_SHORT/UNSIGNED_INT] určuje datový typ indexového pole
indices - indexové pole

void **glDrawRangeElements**(GLenum **mode**, GLuint **start**, GLuint **end**, GLsizei **count**,GLenum **type**,void***indices**);
- dereference omezeného seznamu prvků polí. Omezení je zadáno pomocí start a end indexu. Start ani end nesmí ukazovat mimo indexové pole. Stav není ošetřen.
- jedná se o rozšíření funkce **glDrawElements**.

void **glDrawArrays**(GLenummode, GLintfirst, GLsizei count);
- dereference sekvenčního pole prvků.
- Vytvoří sekvenci geometrických primitiv s použitím prvků polí počínaje first a konče **first**+ **count** –1.

Optimální maximální hodnoty polí je možné zjistit pomocí:
glGetIntegerv(GL_MAX_ELEMENTS_VERTICES/GL_MAX_ELEMENTS_INDICES)

54) Jaký je postup při tessellaci polygonu? Popište funkce gluNewTess(), gluTessCallback(), gluTessProperties(), gluDeleteTess() Co je to winding number?

V některých aplikacích pro tvorbu komplikovaných modelů se používají i složitější polygony. Buď se jedná o nekonvexní polygony, polygony s otvory, nebo o polygony, na kterých se nachází vícenásobné vrcholy (tj. polygony obsahují tzv. degenerované hrany). GLU nabízí nástroj pro převod obecných polygonů do formy vhodné pro vykreslování. Každý polygon musí obsahovat alespoň jednu konturu. Kontura musí tvořit hranici nenulové plochy, tj. musí být zadána alespoň trojicí vrcholů, které v tomto případě nemohou ležet v jedné přímce, ale musí tvořit trojúhelník.

ProcesTessellace
1. Vytvoř nový tessellační objekt pomocí **gluNewTess()**
2. Použij (opakovaně) **gluTessCallback()** k registraci callback funkcí provádějících operace během tesselace (jsou automaticky volány například na začátku teselace, pro zpracování dat vrcholu...)
3. Specifikuj **gluTessProperties()**
4. Vytvoř a vykresli nové tessellované polygony (voláním funkcí **gluTessBeginPolygon**, **gluTessBeginContour**, **gluTessVertex**, ..., **gluTessEndContour**, **gluTessEndPolygon**).
5. Jestliže chcete pokračovat v tessellaci, můžete znovu použít tessellační objekt. Jestliže jste hotovi, můžete jej smazat pomocí **gluDeleteTess()**

Winding pravidla – klasifikují region jakožto vnitřní, jestliže jeho **winding number** patří do zvolené kategorie. Například jestliže je číslo liché, nenulové... .

GLUtesselator* gluNewTess(void**) –** Vytvoří nový tessellační objekt a vrátí ukazatel na něj.
gluDeleteTess(**GLUtesselator***tessobj) – Zruší tessallační objekt.

GluTessCallback(**GLUtesselator***tessobj, **GLenum** type, void (***fn**)()) - registruje CallBack
- fn – specifikuje ukazatel na uživatelskou funkci.
- type – specifikuje typ zpetného volání.

Typy zpětného volání:
GLU_TESS_BEGIN void begin(GLenumtype);
GLU_TESS_BEGIN_DATA void begin(GLenumtype, void *user_data);
GLU_TESS_VERTEX[...] void vertex(void *vertex_data, ...);
GLU_TESS_END[...] void end(void, ...);
GLU_TESS_ERROR[...] void error(GLenum errno);
GLU_TESS_EDGE_FLAG[_DATA] void edgeFlag(GLbooleanflag,...);*
GLU_TESS_COMBINE[...] void combine(GLdouble coords[3], void *vertex_data[4],GLfloatweight[4],void **outData,...);

gluTessProperty(**GLUtesselator***tessobj, **GLenumproperty**, Gldoublevalue);
- může být nastavena pouze jedna vlastnost.

GLU_TESS_BOUNDARY_ONL	True/False
GLU_TESS_TOLERANCE	Distance
GLU_TESS_WINDING_RUL	GLU_TESS_WINDING_ODD (implicitní)
	GLU_TESS_WINDING_NONZERO
	GLU_TESS_WINDING_POZITIVE
	GLU_TESS_WINDING_NEGATIVE
	GLU_TESS_WINDING_ABS_GEQ_TWO

55) Jaký je postup při vytváření kvadrik pomocí knihovny GLU? Popište funkce gluNewQuadric(), gluQuadricOrientation, gluQuadricDrawStyle(), gluQuadricNormals, gluQuadricTexture, gluQuadricCallback. Které kvadriky je možno vykreslit a jak se kreslí?

Kvadrika označuje trojrozměrná tělesa, jejichž povrch (plocha) je určena implicitní funkcí ve tvaru:
f(x, y, z) = a1x2 + a2y2 + a3z2 + a4xy + a5yz + a6xz + a7x + a8y + a9z + a10 = 0
jinými slovy - jedná se, podle nejvyšší použité mocniny, o algebraické plochy druhého stupně, což znamená, že pro parametry implicitní plochy a1-a10 je možné zjistit, zda bod P=[x, y, z] leží, či neleží na ploše kvadriky. Podle znaménka výsledku funkce f(x, y, z) lze dokonce (u uzavřených ploch) zjistit, zda bod leží uvnitř, či vně kvadriky.

GLU poskytuje kvadriky: **koule, kvádr, kruh, kruhová výseč.**

Postup vytváření:
1.Vytvoř nový kvadrikový objekt pomocí **gluNewQadric**()
2.Urči vlastnosti vykreslovaných kvadrik pomocí **gluQuadricOrientation**(), **gluQuadricDrawStyle**(), **gluQuadricNormals**() a **gluQuadricTexture**(),
3.Pomocí **gluQuadricCallback**()zaregistruj callback ke zpracování chyb vzniklých při běhu programu
4.Vytvoř a vykresli nové kvadriky pomocí **gluSphere**(), **gluCylinder**(),**gluDisk**() nebo **gluPartialDisk**()
5.Jestliže chcete pokračovat ve vytváření kvadrik, můžete znovu použít kvadrikový objekt. Jestliže jste hotovi, můžete jej smazat pomocí **gluDeleteQuadric**()

GLUquadricObj* **gluNewQuadric**(void);
-Vytvoří nový kvadrikový objekt a vrátí ukazatel na něj.

void **gluQuadricOrientation**(GLUquadricObj*qobj, Glenum orientation);
- *qobj – ukazatel na daný kvadrikový objekt
orientation – GLU_OUTSIDE, GLU_INSIDE
- Kladný vektor ve směru osy Z je považován za směr „ven“ u kruhu a kruhové výseče

void **gluQuadricDrawStyle**(GLUquadricObj*qobj, GlenumdrawStyle);
- *qobj – ukazatel na daný kvadrikový objekt
drawStyle – GLU_POINT, GLU_LINE, GLU_SILHOUETTE a GLU_FILL.
- GLU_SILHOUETTE - určuje, že základní části jsou vykreslovány stejně jako GLU_LINE, avšak nejsou vykreslovány oddělovací čáry

void **gluQuadricNormals**(GLUquadricObj*qobj, GLenum normals);
*qobj – ukazatel na daný kvadrikový objekt
- normals – GLU_NONE, GLU_FLAT aGLU_SMOOTH

void **gluQuadricTexture**(GLUquadricObj*qobj, Glboolean textureCoords);

*qobj – ukazatel na daný kvadrikový objekt
textureCoords – GL_FALSE aGL_TRUE

void **gluQuadricCallback**(GLUquadricObj*qobj, GLenum which, void (*fn)());
*qobj – ukazatel na daný kvadrikový objekt
which – GLU_ERROR
*fn – ukazatel na uživatelem definovanou funkci.

56) Fragment procesor, vertex procesor - jejich funkce v architektuře OpenGL, použití proměnných, typy proměnných a vzájemná provázanost f. a v. procesoru pomocí proměnných. Uveďte příklady proměnných.

Vertex Processor
-Transformace vrcholů
-Transformace normál a normalizace
-Generování texturových souřadnic
-Transformace texturových souřadnic
-Osvětlení-Nastavení barev materiálu

Proměnné:
Vestavěné uniformní (glFog, gl_ModelViewMatrix...)
Uživatelské uniformní (PoziceOka...)
Vestavěné atributy (gl_Color, gl_Vertex, gl_Normal...)
Uživatelské atributy (Rychlost, Tečna...)
Speciální výstupní (gl_Position, gl_PointSize...)

Fragment Processor
-Operace na interpolovaných
-Přístup k texturám-Aplikace textur
-Fog
-Součty barev

Proměnné:
Vestavěné uniformní (gl_Fog, gl_ModelViewMatrix...)
Uživatelské uniformní (PoziceOka...)
Speciální vstupní (gl_FragCoord, gl_FrontFacing...)
Speciální výstupní (gl_FragColor, gl_FragDepth....)

Procesory mezi sebou komunikuji pomocí vestavěných varying proměnných (gl_FrontColor, gl_SecundaryFrontColor, gl_TexCoord...) a uživatelských varying proměnných (Density...).