Grafiikkamoottorin ohjelmointi – TTV14SP

Janne Koponen

15. syyskuuta 2015

Kurssin sisältö

1. Grafiikkamoottori
2. Grafiikkarajapinnat
3. Matriisilaskentaa $\mathbb{R}^n$:ssa
4. C++ - menetelmiä
5. OpenGL-perusteet
6. Tiedostojen käsittely
7. Spritegrafiikkaa

Aikataulu

- vko 36 Grafiikkarajapinnat: OpenGL, DirectX, OpenGL ES, OpenGL-ikkunan luominen (GLUT, Windows, SDL)
- vko 36 C++ menetelmiä: Nimiavaruudet, dynaaminen muistinhallinta (erityisesti smart-pointer, singleton, assert ja virheiden käsittely, poikkeukset) ja matriisilaskennan kertaus
- vko 37 OpenGL 2D:n perusteet, Grafiikkaprimitiivien piirto (verteksit, indeksit yms)
- vko 38 Tekstuurit, tiedostojen käsittely (objektien lataus ja tallettaminen)
- vko 39 Sprite-grafiikkaa, törmäykset

Aikataulu

vko 36				
	1 - 4	ke	2.9.	8:30 - 12:00
	5 - 8	to	3.9.	13:15 - 16:30
	9 - 12	pe	4.9.	8:30 - 12:00
vko 37				
	13 - 16	ke	9.9.	8:30 - 12:00
	17 - 20	to	10.9.	13:15 - 16:30
	21 - 24	pe	11.9.	8:30 - 12:00
vko 38				
	25 - 28	ke	16.9.	8:30 - 12:00
	29 - 32	to	17.9.	13:15 - 16:30
vko 39				
	33 - 36	ke	23.9.	8:30 - 12:00
	37 - 40	to	24.9.	13:15 - 16:30

Kurssin suorittaminen

- Harjoitustyö – 2D-pelimoottori yhdessä projektiopintojen kanssa.

Arviointi

- Kiitettävä (5): Opiskelija osaa määritellä, suunnitella, toteuttaa ja testata itsenäisesti kompleksisen 2D-grafiikkamoottorin käyttäen OpenGL-grafiikkarajapintaa. Opiskelija osoittaa hyvää ongelmanratkaisukykyä ja osaa itsenäisesti soveltaa taitojaan.
- Hyvä (3-4): Opiskelija osaa määritellä, suunnitella, toteuttaa ja testata itsenäisesti yksinkertaisen 2D-grafiikkamoottorin käyttäen OpenGL-grafiikkarajapintaa. Opiskelija osaa toimia aloitteellisesti ja vastuullisesti annetuissa tehtävissä.
- Tyydyttävä (1-2): Opiskelija osaa määritellä, suunnitella, toteuttaa ja testata ohjatusti yksinkertaisen 2D-grafiikkamoottorin käyttäen OpenGL-grafiikkarajapintaa.

Lähteitä

- Greogory: Game Engine architecture
- Sheiner et al.: OpenGL Programming guide
- `https://open.gl`, `http://www.opengl-tutorial.org/` jne.

Mikä se on?

- Grafiikan käsittelyyn erikoistunut osa pelimoottoria, ei välttämättä tarkkaan määritelty kokonaisuus.
- Objektien piirto näytölle (*rendering*)
- Animointi
- Efektien luonti (sumu, kameraefektit jne.)
- Taustan piirto
- HUD:n piirto
- Tekstien piirto näytölle
- Grafiikkaobjektien lataaminen
- Objektien ajonaikainen muokkaaminen (venytys, peilaus, kierto, luodinreijät jne.)
- Koordinaatistojen hallinta
- Tuki debug-objektien piirtoon (apuviivat yms.)
- Törmäysten havaitseminen
- jne.

3D-grafiikka

Lisäksi 3D-grafiikkamoottorissa yleensä myös:

- Valaistus
- Materiaalien hallinta
- Kameran hallinta
- jne.

Renderöinti on monissa 3D-moottoreissa huomattavasti monimutkaisempi kuin 2D-moottoreissa.

Fysiikkamoottori

- Fysiikkamoottori on pelimoottorin osa, joka huolehtii pelin fysiikkamallinnuksesta.
- Usein raja fysiikka- ja grafiikkamoottorin välillä on häilyvä.
  - Fysiikkaperustaisesti muuttuvat objektit:
  - Vaatteet
  - Hiukset ja muu karvoitus
  - Materiaalien joustaminen (esim. antenni)
  - Nesteet

Tärkeimmät rajapinnat

- OpenGL
  - Silicon Graphics julkaisi 1992
  - Nykyään kehitystä koordinoi Khronos Group
  - Useita alustoja: Windows, Linux, Mac OS X.
  - Suunniteltu yleiskäyttöiseksi 3d-rajapinnaksi.
  - → Paljon turhia ominaisuuksia peliohjelmointiin.
  - Mobiiliversio: OpenGL ES
- DirectX – Direct3D (ja Direct2D)
  - Microsoft 1994
  - Suunniteltu pelikäyttöön.
  - Windows 95 SP2 ja uudemmat& XBox
  - Mobiilikäyttöön Direct3D mobile.
- Molemmat ominaisuuksiltaan hyvin samankaltaisia.

Kolmio DirectX:llä 1/11

```
1  #include <windows.h>
2  #include <windowsx.h>
3  #include <d3d9.h>
4
5  // define the screen resolution
6  #define SCREEN_WIDTH 800
7  #define SCREEN_HEIGHT 600
8
9  // include the Direct3D Library file
10 #pragma comment (lib, "d3d9.lib")
11
12 // global declarations
13 LPDIRECT3D9 d3d;    // ptr to Direct3D interface
14 LPDIRECT3DDEVICE9 d3ddev; // ptr to the device class
15 LPDIRECT3DVERTEXBUFFER9 v_buffer=NULL;
16 //^ptr to the vertex buff
```

Kolmio DirectX:llä 2/11

```
1  void initD3D(HWND hWnd);// sets up and initializes D3D
2  void render_frame(void);// renders a single frame
3  void cleanD3D(void);    // closes D3D and releases memory
4  void init_graphics(void);// 3D declarations
5
6  struct CUSTOMVERTEX {FLOAT X, Y, Z, RHW; DWORD COLOR;};
7  #define CUSTOMFVF (D3DFVF_XYZRHW | D3DFVF_DIFFUSE)
8
9  // the WindowProc function prototype
10 LRESULT CALLBACK WindowProc(HWND hWnd, UINT message,
11                             WPARAM wParam, LPARAM lParam);
```

Kolmio DirectX:llä 3/11

```
1  int WINAPI WinMain(HINSTANCE hInstance,
2                     HINSTANCE hPrevInstance,
3                     LPSTR lpCmdLine,
4                     int nCmdShow)
5  {
6      HWND hWnd;
7      WNDCLASSEX wc;
8
9      ZeroMemory(&wc, sizeof(WNDCLASSEX));
10
11     wc.cbSize = sizeof(WNDCLASSEX);
12     wc.style = CS_HREDRAW | CS_VREDRAW;
13     wc.lpfnWndProc = WindowProc;
14     wc.hInstance = hInstance;
15     wc.hCursor = LoadCursor(NULL, IDC_ARROW);
16     wc.lpszClassName = (LPSTR)L"WindowClass";
17
18     RegisterClassEx(&wc);
```

Kolmio DirectX:llä 4/11

```
1     hWnd = CreateWindowEx(NULL,
2                           (LPSTR)L"WindowClass",
3                           (LPSTR)L"Our Direct3D Program",
4                           WS_OVERLAPPEDWINDOW,
5                           0, 0,
6                           SCREEN_WIDTH, SCREEN_HEIGHT,
7                           NULL,
8                           NULL,
9                           hInstance,
10                          NULL);
11
12    ShowWindow(hWnd, nCmdShow);
13
14    // set up and initialize Direct3D
15    initD3D(hWnd);
16
17    // enter the main loop:
18    MSG msg;
```

Kolmio DirectX:llä 5/11

```
1     while(TRUE)
2     {
3         while(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
4         {
5             TranslateMessage(&msg);
6             DispatchMessage(&msg);
7         }
8
9         if(msg.message == WM_QUIT)
10            break;
11
12        render_frame();
13    }
14
15    // clean up DirectX and COM
16    cleanD3D();
17
18    return msg.wParam;
19 }
```

Kolmio DirectX:llä 6/11

```
1  // this is the main message handler for the program
2  LRESULT CALLBACK WindowProc(HWND hWnd, UINT message, ↩
       WPARAM wParam, LPARAM lParam)
3  {
4    switch(message)
5    {
6      case WM_DESTROY:
7      {
8        PostQuitMessage(0);
9        return 0;
10     } break;
11   }
12
13   return DefWindowProc (hWnd, message, wParam, lParam);
14 }
```

Kolmio DirectX:llä 7/11

```
1  void initD3D(HWND hWnd) {
2    d3d = Direct3DCreate9(D3D_SDK_VERSION);
3
4    D3DPRESENT_PARAMETERS d3dpp;
5
6    ZeroMemory(&d3dpp, sizeof(d3dpp));
7    d3dpp.Windowed = TRUE;
8    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
9    d3dpp.hDeviceWindow = hWnd;
10   d3dpp.BackBufferFormat = D3DFMT_X8R8G8B8;
11   d3dpp.BackBufferWidth = SCREEN_WIDTH;
12   d3dpp.BackBufferHeight = SCREEN_HEIGHT;
13
14   // create a device class
15   d3d->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL,
16   hWnd, D3DCREATE_SOFTWARE_VERTEXPROCESSING,
17   &d3dpp, &d3ddev);
18
19   init_graphics();// initialize the triangle
20 }
```

Kolmio DirectX:llä 8/11

```
1  void render_frame(void)
2  {
3    d3ddev->Clear(0, NULL, D3DCLEAR_TARGET,
4      D3DCOLOR_XRGB(0, 0, 0), 1.0f, 0);
5
6      d3ddev->BeginScene();
7
8      // select which vertex format we are using
9      d3ddev->SetFVF(CUSTOMFVF);
10
11     // select the vertex buffer to display
12     d3ddev->SetStreamSource(0, v_buffer, 0, sizeof(↩
           CUSTOMVERTEX));
13
14     // copy the vertex buffer to the back buffer
15     d3ddev->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 1);
16
17     d3ddev->EndScene();
18
19     d3ddev->Present(NULL, NULL, NULL, NULL);
20 }
```

Kolmio DirectX:llä 9/11

```
1 // this is the function that cleans up Direct3D and COM
2 void cleanD3D(void)
3 {
4     v_buffer->Release();// close and release the vertex ↩
          buffer
5     d3ddev->Release();// close and release the 3D device
6     d3d->Release();// close and release Direct3D
7 }
```

Kolmio DirectX:llä 10/11

```
1  // this function puts the 3D models into video RAM
2  void init_graphics(void)
3  {
4      // create the vertices using the CUSTOMVERTEX struct
5      CUSTOMVERTEX vertices[] =
6      {{400.0f,62.5f,0.5f,1.0f,D3DCOLOR_XRGB(0,0,255),},
7       {650.0f,500.0f,0.5f,1.0f,D3DCOLOR_XRGB(0, 255, 0),},
8       {150.0f,500.0f,0.5f,1.0f,D3DCOLOR_XRGB(255, 0, 0),},
9      };
10
11     // create a vertex buffer interface called v_buffer
12     d3ddev->CreateVertexBuffer(3*sizeof(CUSTOMVERTEX),
13                0, CUSTOMFVF, D3DPOOL_MANAGED,
14               &v_buffer, NULL);
```

Kolmio DirectX:llä 11/11

```
1       VOID* pVoid;
2
3       // lock v_buffer and load the vertices into it
4       v_buffer->Lock(0, 0, (void**)&pVoid, 0);
5       memcpy(pVoid, vertices, sizeof(vertices));
6       v_buffer->Unlock();
7   }
```

Tulos

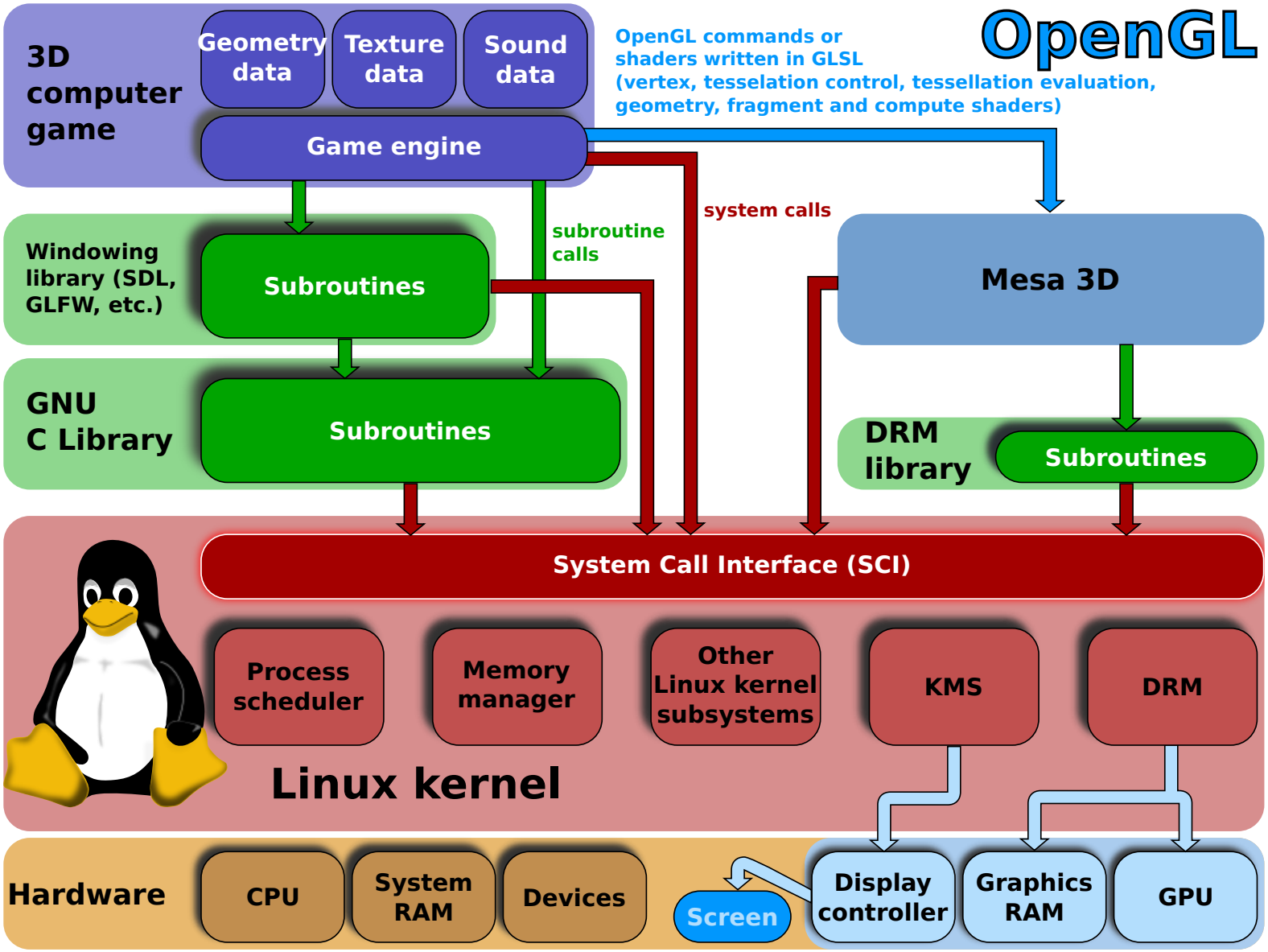
Yleistä

- Erittäin laajalti käytössä
- Crossplatform

→ Grafiikkakutsut ovat yhteensopivia eri järjestelmien kesken mutta muu osa koodista ei välttämättä.

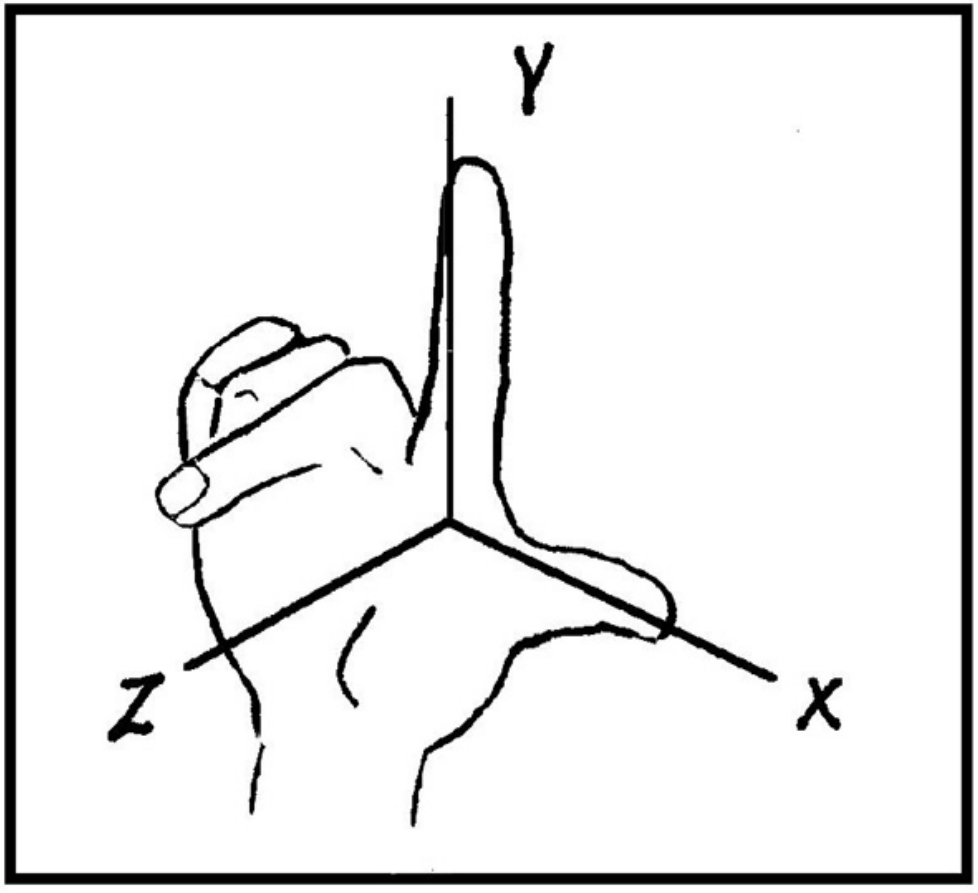
- OpenGL ei tarjoa suoraa tukea ikkunoinnille.
  - Esimerkiksi glut ja GLFW kirjastot on suunniteltu tähän tarkoitukseen. Ne eivät kuitenkaan tarjoa kovin kattavasti muita tarpeellisia ominaisuuksia.
  - Yleensä kannattaa käyttää käyttöjärjestelmän tai jonkin crossplatform-kirjaston ikkunointia kuten esimerkiksi SDL, Qt, FLTK jne.
- Mahdollisuus valmistajakohtaisiin laajennuksiin.

OpenGL pelikäytössä



Oikeakätinen koordinaatisto

- 2D-grafiikassa käytämme oikeakätistä koordinaatistoa.



(Ex 1) Miksi määrittelimme z-akselin 2d-koordinaatistoon?

Kolmio OpenGL:llä 1/2

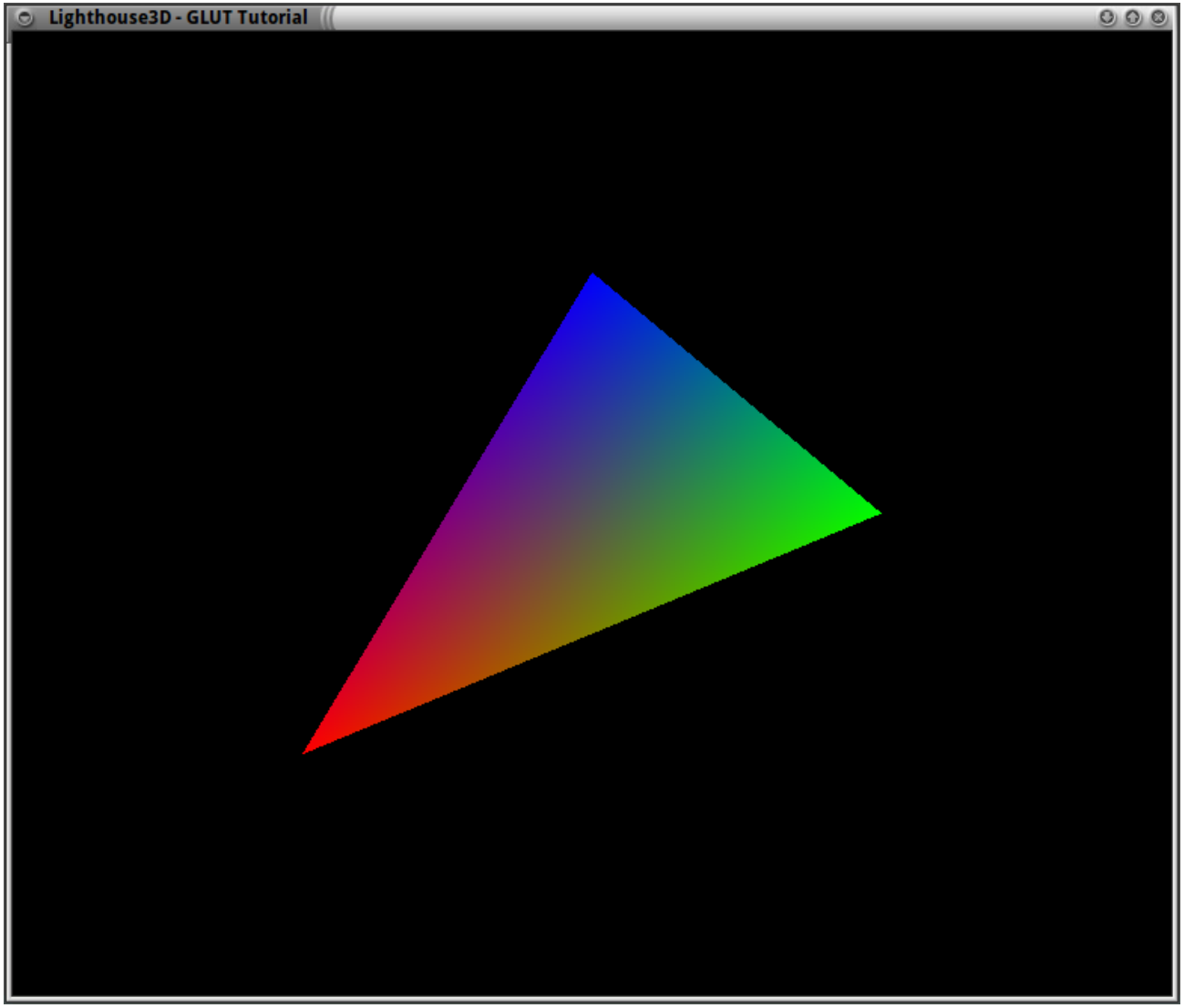
```
1  #include <GL/glut.h>
2
3  void renderScene(void) {
4    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
5
6    glBegin(GL_TRIANGLES);
7    glColor3f(1.0, 0.0, 0.0);
8    glVertex3f(-0.5,-0.5,0.0);
9    glColor3f(0.0, 1.0, 0.0);
10   glVertex3f(0.5,0.0,0.0);
11   glColor3f(0.0, 0.0, 1.0);
12   glVertex3f(0.0,0.5,0.0);
13   glEnd();
14
15   glutSwapBuffers();
16 }
```

Huom: Tämä tapa on tehoton. Oikeasti verteksit kannattaa tallettaa listaan kuten aikaisemmassa Direct3D-esimerkissä.

Kolmio OpenGL:llä 2/2

```
1  int main(int argc, char **argv) {
2
3    // init GLUT and create Window
4    glutInit(&argc, argv);
5    glutInitDisplayMode(GLUT_DEPTH|GLUT_DOUBLE|GLUT_RGBA);
6    glutInitWindowPosition(100,100);
7    glutInitWindowSize(320,320);
8    glutCreateWindow("Lighthouse3D - GLUT Tutorial");
9
10   // register callbacks
11   glutDisplayFunc(renderScene);
12
13   // enter GLUT event processing cycle
14   glutMainLoop();
15
16   return 1;
17 }
```

Tulos



Taustaa

- OpenGL Utility Toolkit
- glut on kirjasto, jonka avulla OpenGL-ohjelmiin saadaan helposti tuki ikkunoinnille.
- `glutMainLoop` ei palauta kontrollia muulle ohjelmalle.
  - Eri toteutuksissa erilaisia tapoja kiertää tämä rajoitus.
  - → Epäyhteensopivia toteutuksia

→ Lähinnä opetus- ja testitarkoituksiin.

- Erittäin helppokäyttöinen näissä tarkoituksissa.
- Windows-ympäristössä glut ei ole oletusarvoisesti asennettuna!
- Dokumentaatio
  `http://freeglut.sourceforge.net/docs/api.php` tai
  `https://www.opengl.org/resources/libraries/glut/spec3/node1.html`

Harjoitus

(Ex 2) Asenna FreeGLUT koneellesi sivulta `http://www.transmissionzero.co.uk/software/freeglut-devel/` löytyvän paketin avulla. Muista hakea MSVC-paketti!

(Ex 3) Käännä ja aja aikaisempi OpenGL-esimerkki.

glut-ohjelman prototyyppi

```
1  #include <GL/glut.h>
2
3  int main(int argc, char **argv) {
4    glutInit(&argc, argv);
5    glutInitDisplayMode(nayttomoodi);
6    glutInitWindowPosition(xPos, yPos);
7    glutInitWindowSize(width, height);
8    glutCreateWindow(windowTitle);
9
10   //Callback-funktioiden rekisterointi
11   //Ainakin tama on syyta olla...
12   glutDisplayFunc(renderCallback);
13
14   glutMainLoop();
15
16   return 1;
17 }
```

Callback-funktiot

- glut kutsuu `glutMainLoop`-funktiosta rekisteröityjä callback-funktioita. Muuta kontrollia ohjelman toimintaan ei ole (paitsi laajennusten kautta).
- Tärkeimmät callback-funktioiden rekisteröintifunktiot:
  - `void glutDisplayFunc(void (*func)(void));`
  - `void glutReshapeFunc(void (*func)(int width, int height));`
  - `void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));` – HUOM: `x` ja `y` käyttävät eri koordinaatistoa kuin piirto
  - `void glutMouseFunc(void (*func)(int button, int state, int x, int y));`
  - `void glutIdleFunc(void (*func)(void));`
  - `void glutTimerFunc(unsigned int msecs, void (*func)(int value), value);`

Callback-funktioiden käyttö 1/3

```
1  #include <GL/glut.h>
2  GLfloat angle=0.0;
3  GLfloat speed=5.0;
4  static int window;
5
6  void renderScene(void) {
7    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
8
9    glLoadIdentity();
10   glRotatef(angle, 0, 0, 1);
11
12   glBegin(GL_TRIANGLES);
13   glColor3f(1.0, 0.0, 0.0); glVertex3f(-0.5,-0.5,0.0);
14   glColor3f(0.0, 1.0, 0.0); glVertex3f(0.5,0.0,0.0);
15   glColor3f(0.0, 0.0, 1.0); glVertex3f(0.0,0.5,0.0);
16   glEnd();
17
18   glutSwapBuffers();
19 }
```

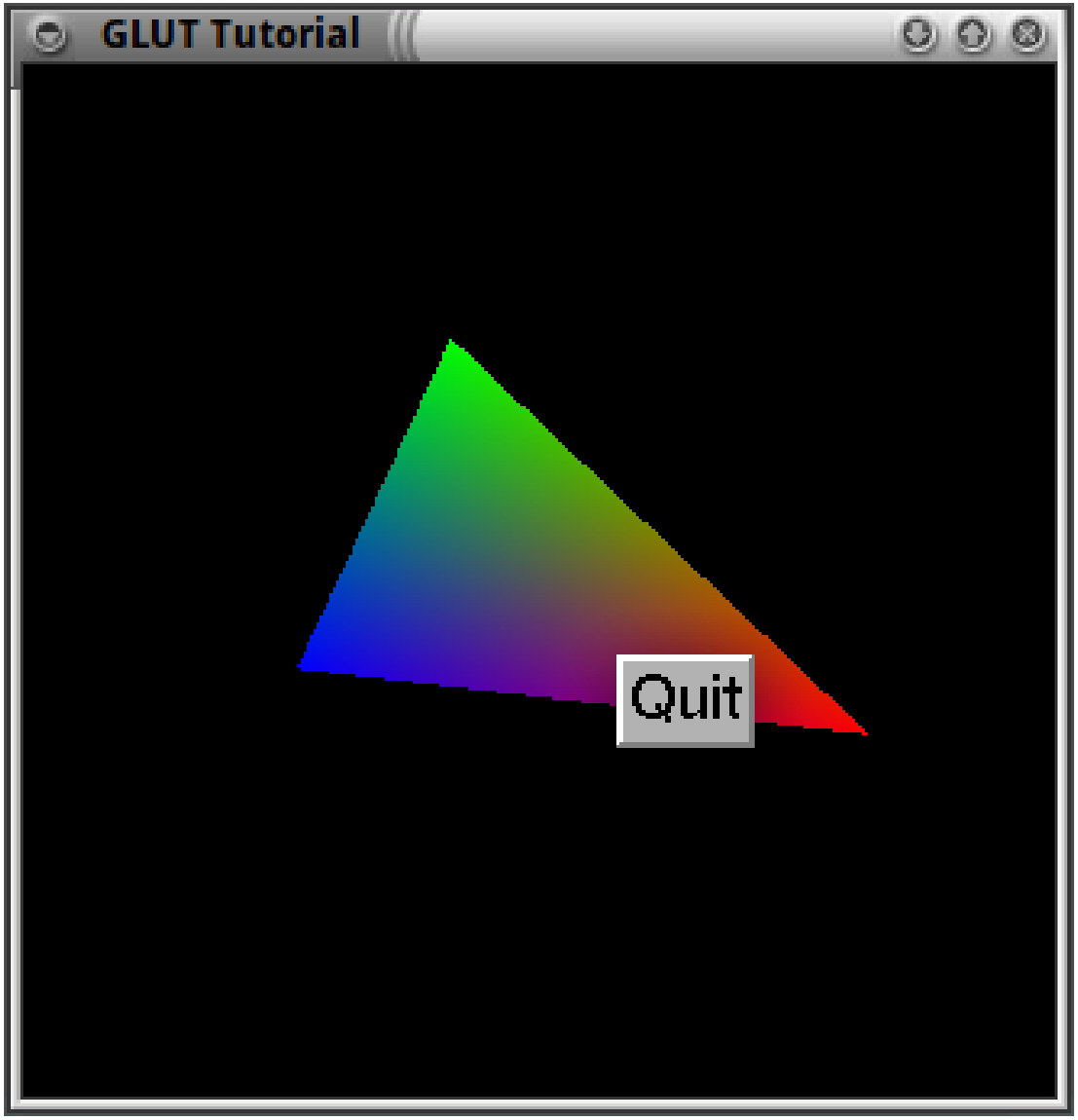
Callback-funktioiden käyttö 2/3

```
1  void timerCallback(int) {
2    angle+=speed;
3    glutTimerFunc(50, timerCallback, 0);
4    renderScene();
5  }
6
7  void kbCallback(unsigned char key, int, int) {
8    speed=-speed;
9  }
10
11 void menuCallback(int entry) {
12   if(entry==1) {
13     glutDestroyWindow(window);
14     exit(0);
15   }
16 }
```

Callback-funktioiden käyttö 3/3

```
1  int main(int argc, char **argv) {
2    glutInit(&argc, argv);
3    glutInitDisplayMode(GLUT_DEPTH|GLUT_DOUBLE|GLUT_RGBA);
4    glutInitWindowPosition(100,100);
5    glutInitWindowSize(320,320);
6    window=glutCreateWindow("GLUT Tutorial");
7
8    glutDisplayFunc(renderScene);
9    glutTimerFunc(50, timerCallback, 0);
10   glutKeyboardFunc(kbCallback);
11
12   glutCreateMenu(menuCallback);
13   glutAddMenuEntry("Quit", 1);
14   glutAttachMenu(GLUT_LEFT_BUTTON);
15
16   glutMainLoop();
17 }
```

Ajoesimerkki



Harjoituksia

(Ex 4) Tee ohjelma joka piirtää näytölle täytetyn n-kulmion, $n \in [3; 8]$. Käyttäjä valitsee valikosta (tulee esiin oikealla napilla) n:n arvon, alkutilanteessa $n = 3$. Alla malli nelikulmion piirrosta:

```
1  glBegin(GL_TRIANGLES);
2  glColor3f(0.5, 0.5, 0.5);
3  glVertex3f(-0.5,-0.5,0.0);glVertex3f(-0.5, 0.5,0.0);
4  glVertex3f( 0.0, 0.0,0.0);
5  glVertex3f(-0.5, 0.5,0.0);glVertex3f( 0.5, 0.5,0.0);
6  glVertex3f( 0.0, 0.0,0.0);
7  glVertex3f( 0.5, 0.5,0.0);glVertex3f( 0.5,-0.5,0.0);
8  glVertex3f( 0.0, 0.0,0.0);
9  glVertex3f( 0.5,-0.5,0.0);glVertex3f(-0.5,-0.5,0.0);
10 glVertex3f( 0.0, 0.0,0.0);
11 glEnd();
```

(Ex 5) Lisää edelliseen ohjelmaan ominaisuus joka vaihtaa n-kulmion väriä satunnaisesti 2 sekunnin välein.

(Ex 6) Tee pohja glut-projektille.

Yleistä

- Windows API:lla ei voi suoraan luoda yhdellä komennolla OpenGL-ikkunaa kuten glut:lla.
- Yksinkertaiset ohjelmat voivat sisältää huomattavasti enemmän koodia kuin glut:lla työskennellessä.
- Vastaavasti koko Windows API:n tarjoama joustavuus on käytössä myös OpenGL-ohjelmissa.
  - Glut-ohjelmien rajoittuneisuus johtuu glut:n arkkitehtuurista.
  - Ei ole mitään syytä miksi OpenGL-ohjelmien tulisi olla ominaisuuksiltaan rajoittuneempia kuin muiden ohjelmien.
- Myös MFC:llä on mahdollista tehdä OpenGL-ohjelmistoja. Katso esimerkiksi `https://msdn.microsoft.com/en-us/library/7zx6z2e4%28v=vs.90%29.aspx`.

Kolmio Windowsilla 1/17

```
1  #include <windows.h>
2  #include <GL/gl.h>
3
4  const char *className = "OpenGL";
5  const char *windowName = "OpenGL Triangle";
6  int winX = 0, winY = 0;
7  int winWidth = 400, winHeight = 400;
8  HDC hDC; //Device context
9  HGLRC hGLRC; //Rendering context
10 HPALETTE hPalette;
11
12 void init(void) {
13   glClearColor( 0, 0, 0, 0 );
14
15   glViewport( 0, 0, winWidth, winHeight );
16
17   glMatrixMode( GL_PROJECTION );
18   glLoadIdentity( );
19 }
```

Kolmio Windowsilla 2/17

```
1  void redraw(void) {
2    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
3
4    glBegin(GL_TRIANGLES);
5    glColor3f(1.0, 0.0, 0.0);
6    glVertex3f(-0.5,-0.5,0.0);
7    glColor3f(0.0, 1.0, 0.0);
8    glVertex3f(0.5,0.0,0.0);
9    glColor3f(0.0, 0.0, 1.0);
10   glVertex3f(0.0,0.5,0.0);
11   glEnd();
12
13   SwapBuffers(hDC);
14 }
15
16 void resize(void) {
17   glViewport(0, 0, winWidth, winHeight);
18 }
```

Kolmio Windowsilla 3/17

```
1  void setupPixelFormat(HDC hDC) {
2    PIXELFORMATDESCRIPTOR pfd = {
3      sizeof(PIXELFORMATDESCRIPTOR), /* size */
4      1,                             /* version */
5      PFD_SUPPORT_OPENGL |
6      PFD_DRAW_TO_WINDOW |
7      PFD_DOUBLEBUFFER,   /* support double-buffering */
8      PFD_TYPE_RGBA,      /* color type */
9      16,                 /* prefered color depth */
10     0, 0, 0, 0, 0, 0,   /* color bits (ignored) */
11     0,                  /* no alpha buffer */
12     0,                  /* alpha bits (ignored) */
13     0,                  /* no accumulation buffer */
14     0, 0, 0, 0,         /* accum bits (ignored) */
15     16,                 /* depth buffer */
16     0,                  /* no stencil buffer */
17     0,                  /* no auxiliary buffers */
18     PFD_MAIN_PLANE,     /* main layer */
19     0,                  /* reserved */
20     0, 0, 0,         /* no layer, visible, damage masks */
21   };
```

Kolmio Windowsilla 4/17

```
1   int pixelFormat = ChoosePixelFormat(hDC, &pfd);
2   if (pixelFormat == 0) {
3     MessageBox(WindowFromDC(hDC),
4         "ChoosePixelFormat failed.",
5         "Error", MB_ICONERROR | MB_OK);
6     exit(1);
7   }
8
9   if (SetPixelFormat(hDC, pixelFormat, &pfd) != TRUE) {
10    MessageBox(WindowFromDC(hDC),
11        "SetPixelFormat failed.", "Error",
12 MB_ICONERROR | MB_OK);
13    exit(1);
14  }
15 }
```

Kolmio Windowsilla 5/17

```
1  void setupPalette(HDC hDC) {
2    int pixelFormat = GetPixelFormat(hDC);
3    PIXELFORMATDESCRIPTOR pfd;
4    LOGPALETTE* pPal;
5    int paletteSize;
6
7    DescribePixelFormat(hDC, pixelFormat,
8          sizeof(PIXELFORMATDESCRIPTOR), &pfd);
9
10   if (pfd.dwFlags & PFD_NEED_PALETTE) {
11     paletteSize = (1 << pfd.cColorBits);
12   } else {
13     return;
14   }
15
16   pPal = (LOGPALETTE*)
17   malloc(sizeof(LOGPALETTE) +
18        paletteSize * sizeof(PALETTEENTRY));
19   pPal->palVersion = 0x300;
20   pPal->palNumEntries = paletteSize;
```

Kolmio Windowsilla 6/17

```
1   /* build a simple RGB color palette */
2   int redMask = (1 << pfd.cRedBits) - 1;
3   int greenMask = (1 << pfd.cGreenBits) - 1;
4   int blueMask = (1 << pfd.cBlueBits) - 1;
5
6   for (int i=0; i<paletteSize; ++i) {
7     pPal->palPalEntry[i].peRed =
8       (((i>>pfd.cRedShift) & redMask)*255)/redMask;
9     pPal->palPalEntry[i].peGreen =
10      (((i>>pfd.cGreenShift) & greenMask)*255)/greenMask;
11    pPal->palPalEntry[i].peBlue =
12      (((i>>pfd.cBlueShift) & blueMask)*255)/blueMask;
13    pPal->palPalEntry[i].peFlags = 0;
14  }
```

Kolmio Windowsilla 7/17

```
1    hPalette = CreatePalette(pPal);
2    free(pPal);
3
4    if (hPalette) {
5      SelectPalette(hDC, hPalette, FALSE);
6      RealizePalette(hDC);
7    }
8  }
```

Kolmio Windowsilla 8/17

```
1  LRESULT APIENTRY WndProc(
2   HWND hWnd,
3   UINT message,
4   WPARAM wParam,
5   LPARAM lParam)
6  {
7    switch (message) {
8    case WM_CREATE:
9      /* initialize OpenGL rendering */
10     hDC = GetDC(hWnd);
11     setupPixelFormat(hDC);
12     setupPalette(hDC);
13     hGLRC = wglCreateContext(hDC);
14     wglMakeCurrent(hDC, hGLRC);
15     init();
16     return 0;
```

Kolmio Windowsilla 9/17

```
1   case WM_DESTROY:
2     /* finish OpenGL rendering */
3     if (hGLRC) {
4       wglMakeCurrent(NULL, NULL);
5       wglDeleteContext(hGLRC);
6     }
7     if (hPalette) {
8       DeleteObject(hPalette);
9     }
10    ReleaseDC(hWnd, hDC);
11    PostQuitMessage(0);
12    return 0;
```

Kolmio Windowsilla 10/17

```
1   case WM_SIZE:
2     /* track window size changes */
3     if (hGLRC) {
4       winWidth = (int) LOWORD(lParam);
5       winHeight = (int) HIWORD(lParam);
6       resize();
7       return 0;
8     }
```

Kolmio Windowsilla 11/17

```
1   case WM_PALETTECHANGED:
2   /* realize palette if this is *not* the current window ↩
        */
3     if (hGLRC && hPalette && (HWND) wParam != hWnd) {
4       UnrealizeObject(hPalette);
5       SelectPalette(hDC, hPalette, FALSE);
6       RealizePalette(hDC);
7       redraw();
8       break;
9     }
10    break;
```

Kolmio Windowsilla 12/17

```
1   case WM_QUERYNEWPALETTE:
2     /* realize palette if this is the current window */
3     if (hGLRC && hPalette) {
4       UnrealizeObject(hPalette);
5       SelectPalette(hDC, hPalette, FALSE);
6       RealizePalette(hDC);
7       redraw();
8       return TRUE;
9     }
10    break;
```

Kolmio Windowsilla 13/17

```
1   case WM_PAINT:
2     {
3       PAINTSTRUCT ps;
4       BeginPaint(hWnd, &ps);
5       if (hGLRC) {
6         redraw();
7       }
8       EndPaint(hWnd, &ps);
9       return 0;
10    }
11    break;
```

Kolmio Windowsilla 14/17

```
1   case WM_CHAR:
2     /* handle keyboard input */
3     switch ((int)wParam) {
4     case VK_ESCAPE:
5       DestroyWindow(hWnd);
6       return 0;
7     default:
8       break;
9     }
10    break;
11  default:
12    break;
13  }
14  return DefWindowProc(hWnd, message, wParam, lParam);
15 }
```

Kolmio Windowsilla 15/17

```
1 int APIENTRY WinMain(
2       HINSTANCE hCurrentInst,
3       HINSTANCE hPreviousInst,
4       LPSTR lpszCmdLine,
5       int nCmdShow)
6 {
7   WNDCLASS wndClass;
8   HWND hWnd;
9   MSG msg;
```

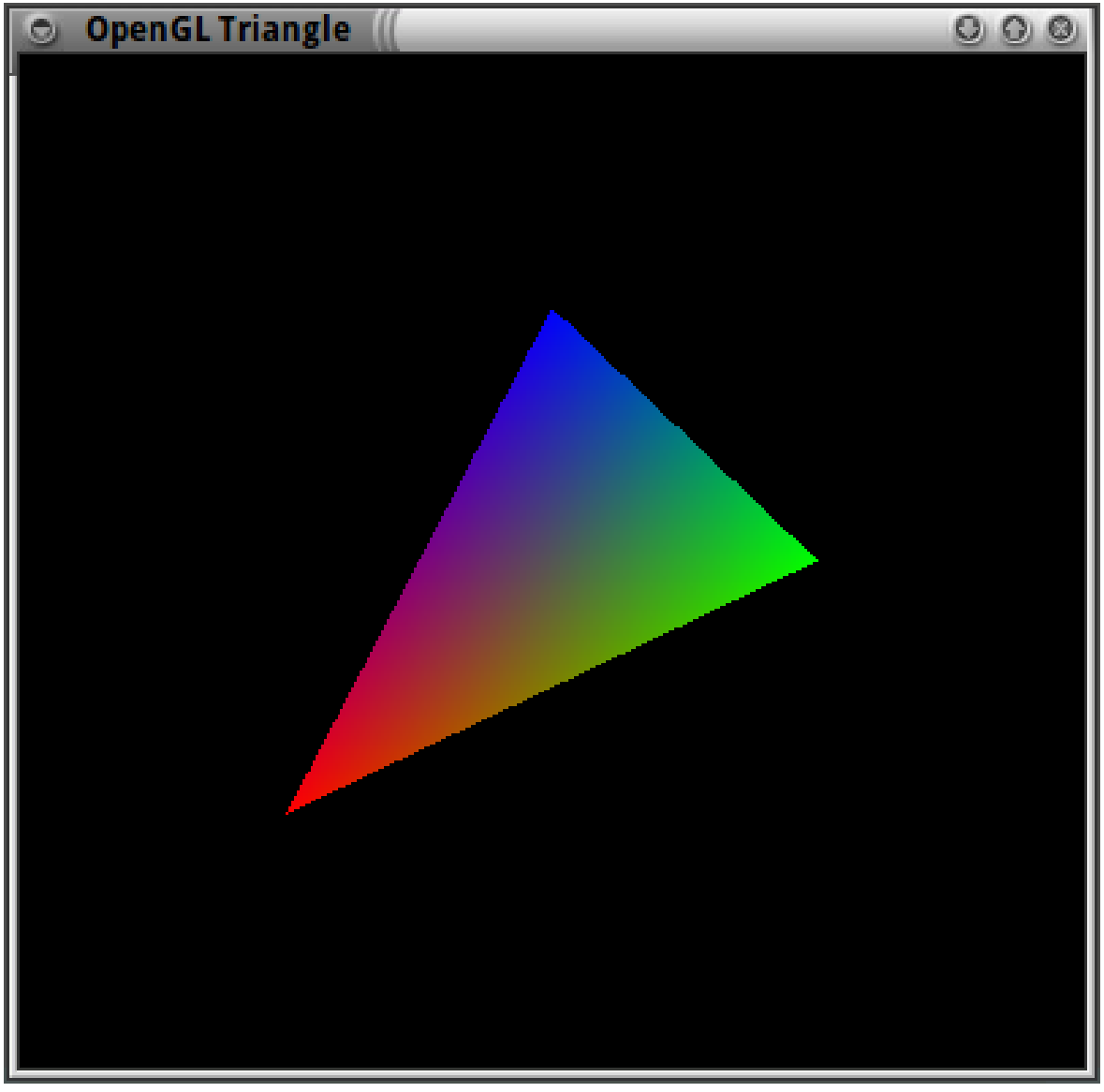
Kolmio Windowsilla 16/17

```
1   /* register window class */
2   wndClass.style = CS_OWNDC | CS_HREDRAW | CS_VREDRAW;
3   wndClass.lpfnWndProc = WndProc;
4   wndClass.cbClsExtra = 0;
5   wndClass.cbWndExtra = 0;
6   wndClass.hInstance = hCurrentInst;
7   wndClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
8   wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
9   wndClass.hbrBackground=
10      (HBRUSH)GetStockObject(BLACK_BRUSH);
11  wndClass.lpszMenuName = NULL;
12  wndClass.lpszClassName = className;
13  RegisterClass(&wndClass);
14
15  /* create window */
16  hWnd = CreateWindow(className, windowName,
17     WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN
18       | WS_CLIPSIBLINGS,
19     winX, winY, winWidth, winHeight,
20     NULL, NULL, hCurrentInst, NULL);
```

Kolmio Windowsilla 17/17

```
1   /* display window */
2   ShowWindow(hWnd, nCmdShow);
3   UpdateWindow(hWnd);
4
5   /* process messages */
6   while (GetMessage(&msg, NULL, 0, 0) == TRUE) {
7     TranslateMessage(&msg);
8     DispatchMessage(&msg);
9   }
10  return msg.wParam;
11 }
```

Ajoesimerkki



Simple DirectMedia Layer (SDL)

- Cross-platform: Linux, Windows, OS X, Android jne.
- Avoimen lähdekoodin kirjasto (zlib License)

→ Mahdollistaa käytön kaupallisissa sovelluksissa

- Laajalti käytetty – Kirjaston www-sivuilta löytyy esimerkiksi yli 700 avoimen lähdekoodin peliä jotka käyttävät SDL:ää.

→ Hyvin testattua koodia

- Jaettu alisysteemeihin: Basics, Video, Input events, Audio, Threads, Timers jne.
- Lisäksi korkeamman tason kirjastoja: SDL_image, SDL_mixer, SDL_net, SDL_ttf ja SDL_rtf.
- Myös suuri määrä “epävirallisia” kirjastoja saatavilla.
- (Esimerkit kirjoitettu versiolla 1.2. Uusin saatavilla oleva versio on 2.0, pahoittelut!)

OpenGL + SDL 1/8

```
1  #include <SDL/SDL.h>
2  #include <GL/gl.h>
3  #include <GL/glu.h>
4
5  #include <stdio.h>
6  #include <stdlib.h>
7
8  static void quit_tutorial( int code )
9  {
10   SDL_Quit( );
11   exit( code );
12 }
```

OpenGL + SDL 2/8

```
1  static void handle_key_down( SDL_keysym* keysym )
2  {
3    switch( keysym->sym ) {
4    case SDLK_ESCAPE:
5      quit_tutorial( 0 );
6      break;
7    default:
8      break;
9    }
10 }
```

OpenGL + SDL 3/8

```
1  static void process_events( void )
2  {
3    SDL_Event event;
4
5    while( SDL_PollEvent( &event ) ) {
6
7      switch( event.type ) {
8        case SDL_KEYDOWN:
9        handle_key_down( &event.key.keysym );
10       break;
11       case SDL_QUIT:
12       quit_tutorial( 0 );
13       break;
14     }
15   }
16 }
```

OpenGL + SDL 4/8

```
1  static void draw_screen( void )
2  {
3    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
4
5    glBegin(GL_TRIANGLES);
6    glColor3f(1.0, 0.0, 0.0);
7    glVertex3f(-0.5,-0.5,0.0);
8    glColor3f(0.0, 1.0, 0.0);
9    glVertex3f(0.5,0.0,0.0);
10   glColor3f(0.0, 0.0, 1.0);
11   glVertex3f(0.0,0.5,0.0);
12   glEnd();
13
14   SDL_GL_SwapBuffers( );
15 }
```

OpenGL + SDL 5/8

```
1  static void setup_opengl( int width, int height )
2  {
3    /* Set the clear color. */
4    glClearColor( 0, 0, 0, 0 );
5
6    /* Setup our viewport. */
7    glViewport( 0, 0, width, height );
8
9    glMatrixMode( GL_PROJECTION );
10   glLoadIdentity( );
11 }
```

OpenGL + SDL 6/8

```
1  int main( int argc, char* argv[] )
2  {
3    const SDL_VideoInfo* info = NULL;
4    int width = 0;
5    int height = 0;
6    int bpp = 0;
7    int flags = 0;
8
9    if( SDL_Init( SDL_INIT_VIDEO ) < 0 ) {
10     fprintf( stderr,
11     "Video initialization failed: %s\n",
12     SDL_GetError( ) );
13     quit_tutorial( 1 );
14   }
15
16   info = SDL_GetVideoInfo( );
17
18   width = 640;
19   height = 480;
20   bpp = info->vfmt->BitsPerPixel;
```

OpenGL + SDL 7/8

```
1  SDL_GL_SetAttribute(SDL_GL_RED_SIZE, 5);//at least 5bpp
2  SDL_GL_SetAttribute( SDL_GL_GREEN_SIZE, 5 );
3  SDL_GL_SetAttribute( SDL_GL_BLUE_SIZE, 5 );
4  SDL_GL_SetAttribute( SDL_GL_DEPTH_SIZE, 16 );
5  SDL_GL_SetAttribute( SDL_GL_DOUBLEBUFFER, 1 );
6
7  flags = SDL_OPENGL;
8
9  if( SDL_SetVideoMode( width, height, bpp, flags )==0) {
10   fprintf( stderr, "Video mode set failed: %s\n",
11            SDL_GetError( ) );
12   quit_tutorial( 1 );
13 }
14
15 setup_opengl( width, height );
```

OpenGL + SDL 8/8

```
1   while( 1 ) {
2     process_events( );
3     draw_screen( );
4   }
5
6   /* Never reached. */
7   return 0;
8 }
```

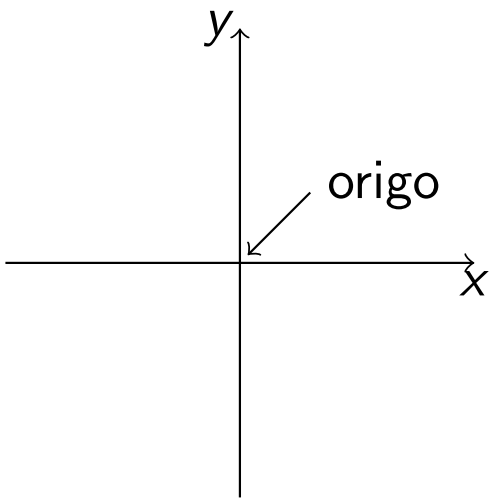
Harjoituksia

(Ex 7) Tee pohja SDL-pohjaiselle OpenGL-projektille.

(Ex 8) Toteuta SDL-pohjainen OpenGL-ohjelma, jossa on pyörivä kolmio ja näppäimellä/hiirellä pyörimissuunta vaihtuu. (Kuten aikaisempi glut-esimerkki)

2-ulotteinen koordinaatisto

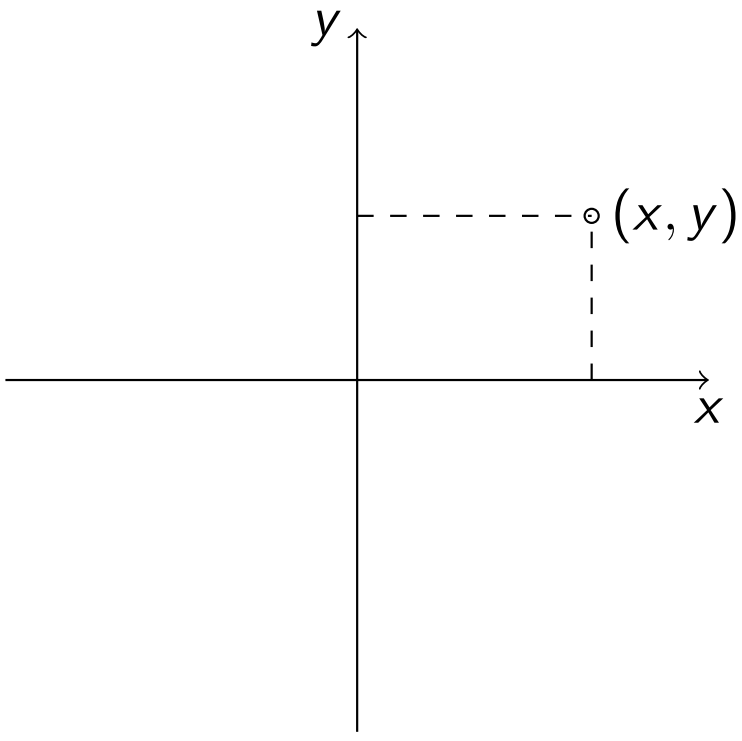
- **Tällä kurssilla esitetään vain kevyt johdatus vektoreihin ja matriiseihin. Samalla matemaattisesta tarkkuudesta tingitään jossain määrin!**
- Tason (2-ulotteisen avaruuden) koordinaatisto muodostetaan valitsemalla yksi tason piste origoksi ja kiinnittämällä siihen kaksi toisiaan vastaan kohtisuoraa koordinaattiakselia.
- Koordinaattiakseleille täytyy kiinnittää positiiviset suunnat.
  - Yleensä käytetään koordinaatistoa, jossa positiiviselta x-akselilta kierryttäessä vastapäivään 90° tullaan positiiviselle y-akselille.



Tason ja $\mathbb{R}^2$:n samaistus 1/2

- Jokaisen tason pisteen paikka voidaan ilmaista lukuparina (x, y).
- Tässä
  - ensimmäinen luku (x) ilmaisee pisteen x-koordinaatin ja
  - jälkimmäinen luku (y) ilmaisee pisteen y-koordinaatin.
- Samaistetaan taso $\mathbb{R}^2$:n kanssa:
  - Jokaista tason pistettä vastaa lukupari (x, y) ja
  - jokaista lukuparia (x, y) vastaa yksi tason piste.

Tason ja $\mathbb{R}^2$:n samaistus 2/2



Taustaa

Esimerkki:

- Pelihahmo istuu autossa joka liikkuu.
- Kuinka pelihahmon vasen käsi saadaan piirrettyä näytölle?
- Käytetään erilaisia koordinaatistoja apuna.
- Tehdään koordinaatistomuunnokset hierarkisesti:
  - Vasemman käden koordinaatisto $\rightarrow$ pelaajan koordinaatisto $\rightarrow$ auton koordinaatisto $\rightarrow$ maailman koordinaatisto $\rightarrow$ kameran koordinaatisto $\rightarrow$ näytön koordinaatisto

Toteutus

- Ideana on laskea kaikille vasemman käden piirtämiseen tarvittaville pisteille yksi kuvaus, jolla kaikki vasemman käden pisteet voidaan muuntaa maailman koordinaatistoon.
- Voidaan luoda puumainen esitys eri koordinaatistoista.
  - Puun solmuun on tallennettu muunnos kyseisestä koordinaatistosta johonkin puun juurta lähempänä olevaan koordinaatistoon (yleensä kameran koordinaatistoon).
  - Muunnos koordinaatistosta toiseen sisältää yleensä kierron, siirroksen ja skaalauksen tai vain joitakin näistä.

Laskenta

- Koordinaatistomuunnosten täsmällinen käsittely vaatii vektoreiden ja matriisien käsitteet.
- Kaikki koordinaatistomuunnokset kuvataan matriiseilla, jotka sitten kerrotaan keskenään, jolloin saadaan yksi kokonaismuunnos.

Muunnetaan esimerkiksi piste **p** toiseen koordinaatistoon. Alkuperäistä pistettä on kierretty muunnoksella **R** ja sen jälkeen siirretty muunnoksella **T**. Tällöin **p** kuvautuu uuteen koordinaatistoon paikkaan **TRp**.

Matriisit

- Samaistetaan avaruuden $\mathbb{R}^n$ piste x (n-jonon) ja vektori $\mathbf{x}$.
- Tämän vektoriavaruuden lineaarikuvaukset voidaan esittää $\mathbb{R}^{n \times n}$ matriiseina.
  - $c\mathbf{A}(\mathbf{x}) = \mathbf{A}(c\mathbf{x})$, $C \in \mathbb{R}$ ja
  - $\mathbf{A}(\mathbf{x}) + \mathbf{A}(\mathbf{y}) = \mathbf{A}(\mathbf{x} + \mathbf{y})$

Matriisin ja vektorin välinen kertolasku

Olkoon $\mathbf{A} \in \mathbb{R}^{m \times n}$ ja $\mathbf{x} \in \mathbb{R}^n$. Määritellään tällöin tulo

$$\mathbf{Ax} = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} a_{1,i} x_i \\ \sum_{i=1}^{n} a_{2,i} x_i \\ \vdots \\ \sum_{i=1}^{n} a_{m,i} x_i \end{pmatrix}.$$

Tulosvektorin alkio paikassa i on

$$(Ax)_i = \begin{pmatrix} a_{i,1} \\ a_{i,2} \\ \vdots \\ a_{i,n} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}.$$

Huomaa, että tässä määriteltiin **Ax**, mutta ei **xA**!

Matriisien kertolasku (1/2)

Edellä esitetty matriisin ja vektorin välinen tulo saadaan erikoistapauksena matriisien välisestä kertolaskusta.
Olkoon $\mathbf{A} \in \mathbb{R}^{m \times n}$ ja $\mathbf{B} \in \mathbb{R}^{n \times p}$. Määritellään tällöin tulo

$$\begin{aligned}
\mathbf{AB} &= \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{pmatrix} \begin{pmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,p} \\ b_{2,1} & b_{2,2} & \dots & b_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \dots & b_{n,p} \end{pmatrix} \\
&= \begin{pmatrix} \sum_{i=1}^{n} a_{1,i} b_{i,1} & \sum_{i=1}^{n} a_{1,i} b_{i,2} & \dots & \sum_{i=1}^{n} a_{1,i} b_{i,p} \\ \sum_{i=1}^{n} a_{2,i} b_{i,1} & \sum_{i=1}^{n} a_{2,i} b_{i,2} & \dots & \sum_{i=1}^{n} a_{2,i} b_{i,p} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} a_{m,i} b_{i,1} & \sum_{i=1}^{n} a_{m,i} b_{i,2} & \dots & \sum_{i=1}^{n} a_{m,i} b_{i,p} \end{pmatrix}.
\end{aligned}$$

Matriisien kertolasku (2/2)

Määritelmästä seuraa suoraan, että $\mathbf{AB} \in \mathbb{R}^{m \times p}$, eli tulomatriisin rivien lukumäärän määrää **A**:n rivien lukumäärä ja sarakkeiden lukumäärän määrää **B**:n sarakkeiden lukumäärä. Lisäksi on huomioitava, että **A**:ssa on oltava yhtä monta saraketta kuin **B**:ssä on rivejä.
Tulomatriisin riville i, sarakkeeseen j tulee kertolaskussa arvo

$$(\mathbf{AB})_{ij} = \begin{pmatrix} a_{i,1} \\ a_{i,2} \\ \vdots \\ a_{i,n} \end{pmatrix} \cdot \begin{pmatrix} b_{1,j} \\ b_{2,j} \\ \vdots \\ b_{n,j} \end{pmatrix} = \sum_{k=1}^{n} a_{i,k} b_{k,j} \ .$$

Huomaa, että yleensä ei päde $\mathbf{AB} = \mathbf{BA}$.

Harjoituksia

(Ex 9) Kirjoita luokka 3×3-matriisien ja 3-vektorien käsittelyyn. **Tätä luokkaa käytetään vain harjoituksissa, joten älä kirjoita "liian hienoa" luokkaa.**

Homogeeninen koordinaatisto

Ongelma: Lineaarikuvaukset kuvaavat origon origoksi $\Rightarrow$ Siirtoa ei voi kuvata lineaarikuvauksella.

Ratkaisu: Valitaan käytettävä avaruus siten, että origo ei kuulu kyseiseen avaruuteen.

1. Upotetaan $\mathbb{R}^2$ $\mathbb{R}^3$:een siten, että asetetaan kolmas koordinaatti aina arvoon 1.
2. Upotetaan $\mathbb{R}^3$ $\mathbb{R}^4$:ään siten, että asetetaan neljäs koordinaatti aina arvoon 1.

Tällä "tempulla" voidaan myös siirrot kuvata lineaarikuvauksella.
Esimerkkejä $\mathbb{R}^2$:ssa

$$\begin{aligned} (0,0) &\sim (0,0,1)^T \\ (x_1,x_2) &\sim (x_1,x_2,1)^T \end{aligned}$$

Siirto $\mathbb{R}^2$:ssa

Määritellään siirtomatriisi

$$\mathbf{T}(\Delta x, \Delta y) = \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix} .$$

Matriisi riippuu siis kahdesta parametrista Δx ja Δy.

Kierto $\mathbb{R}^2$:ssa

Vektoria kierretään origon ympäri kulman θ verran kiertomatriisilla

$$\mathbf{R}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} .$$

Kiertomatriisi $\mathbf{R}(\theta)$ on ortogonaalinen, joten sen käänteismatriisi on $\mathbf{R}(\theta)^T$.

Skaalaus $\mathbb{R}^2$:ssa

Määritellään skaalausmatriisi

$$\mathbf{S}(k_x, k_y) = \begin{pmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Jossa k_x on skaalaus x-akselin suunnassa ja k_y y-akselin suunnassa. Mikäli skaalaus on 0 jonkin akselin suuntaan, saadaan kaikki pisteet projisoitua toiselle koordinaattiakselille. Mikäli skaalauskerroin on negatiivinen, saadaan aikaan peilaus toisen koordinaattiakselin suhteen.

Esimerkki

Karusellin kyydissä istuu henkilö, jonka koordinaatit karusellin koordinaatistossa ovat $\mathbf{p}_0 = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$. Karuselli sijaitsee maailman koordinaatistossa kohdassa $\begin{pmatrix} x \\ y \end{pmatrix}$. Lisäksi karuselli on pyörähtänyt oman origonsa ympäri myötäpäivään kulman α verran. Laske $\mathbf{p}$ =henkilön sijainti maailmankoordinaatistossa.

Ratkaisu:

$$\begin{aligned} \mathbf{p} &= \mathbf{T}(x,y)\mathbf{R}(\alpha)\mathbf{p}_0 \\ &= \begin{pmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ 1 \end{pmatrix} . \end{aligned}$$

(Ex 10) Laske kokonaismuunnoksen matriisi.

Harjoituksia

(Ex 11) Muokkaa glut-esimerkkiä siten että poistat siitä `glRotatef`-funktion ja toteutat kierron itse toteuttamillasi matriiseilla.

(Ex 12) Lisää edelliseen siirto s.e pyörimisen keskipiste on pisteessä $(-0.25, 0.25)$.

(Ex 13) Lisää edelliseen piirrettävän kolmion skaalaaminen x-akselin suunnassa kertoimella 1.25 ja y-akselin suunnassa kertoimella 0.5.

HUOM: Älä toteuta jatkossa koordinaatistomuunnoksia näin, vaan käytä esimerkikis OpenGL:n tai GLM:n tarjoamia valmiita matriiseja!

Edistyksellisiä C++ - menetelmiä

- Nimiavaruudet
  - Tapa välttää päällekkäiset nimet.
  - Käytännöllinen erityisesti isoissa projekteissa tai kirjastoissa.
- Smart-pointer
  - Osoitintyyppi jonka avulla dynaamisesti varatut oliot saadaan tuhottua automaattisesti.
  - “Roskien keruu”
- Singleton-tyyppi
  - Luokka jonka olioita voi luoda vain yhden kappaleen
- Virheiden käsittely
  - `assert`
  - Poikkeukset

Perusteet

- Nimiavaruuden sisään rajataan tyypillisesti kaikki nimiavaruuden käsittämän kokonaisuuden sisältämät tunnisteet (muuttujien, funktioiden, luokkien jne nimet).
- Organisoi koodin loogisiksi kokonaisuuksiksi, joiden tunnisteet eivät "häiritse" toisiaan.

→ Mahdollistaa samojen tunnisteiden käytön useampaan kertaan.

- Esimerkiksi nimiavaruus `std` on C++:n standardikirjastojen käyttämä nimiavaruus.

Tunniste nimiavaruuden sisällä 1/2

- Käytetään tunnisteen kokonaista (fully qualified) nimeä:

```
1 #include <iostream>
2
3 int main(int argc, char **argv) {
4   std::cout<<"Hello world!\n";
5 }
```

- Määritellään käytettäväksi nimiavaruutta `std` `using namespace`-direktiivillä:

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main(int argc, char **argv) {
6   cout<<"Hello world!\n";
7 }
```

Tunniste nimiavaruuden sisällä 2/2

- Määritellään käytettäväksi yhtä tynnistetta nimiavaruuden sisältä:

```
1 #include <iostream>
2
3 using std::cout;
4
5 int main(int argc, char **argv) {
6   cout<<"Hello world!\n";
7 }
```

- **Otsikkotiedostoissa (header) on aina käytettävä kokonaista nimeä!** Miksi?

using-direktiivin vaikutusalue

```
1  #include <iostream>
2
3  //Maaritellaan nimiavaruus
4  namespace foobar {
5    void f(){std::cout<<"f()\n";}
6  }
7
8  int main(int argc, char **argv) {
9    {
10     using namespace foobar;
11     f();
12   }
13
14   //using maaritelty lohkossa, ei vaikuta ulkopuolella
15   foobar::f();
16 }
```

Nimiavaruuteen voi lisätä tunnisteita “monessa erässä”

```
1  #include <iostream>
2
3  namespace foobar {
4    void f(){std::cout<<"f()\n";}
5  }
6
7  namespace foobar {
8    void g();
9  }
10
11 void foobar::g() {std::cout<<"g()\n";}
12
13 int main(int argc, char **argv) {
14     using namespace foobar;
15     f();
16     g();
17 }
```

Nimiavaruuden sisällä ei tarvitse käyttää kokonaisia nimiä tai using-direktiiviä

```
1  #include <iostream>
2
3  namespace foobar {
4    void f(){std::cout<<"f()\n";}
5    void g(){f();}
6  }
7
8  int main(int argc, char **argv) {
9      using namespace foobar;
10     f();
11     g();
12 }
```

Toteutuksen yhteydessä tunnisteille on käytettävä kokonaisia nimiä

```
1  #include <iostream>
2
3  namespace foobar {
4    void f();
5    void g(){f();}
6  }
7
8  using namespace foobar;
9
10 //Tassa on kaytettava aina nimiavaruuden nimea
11 void foobar::f(){std::cout<<"f()\n";}
12
13 int main(int argc, char **argv) {
14     f();
15     g();
16 }
```

Sisäkkäiset nimiavaruudet

```
1  #include <iostream>
2
3  namespace foo {
4    void f();
5    namespace bar {
6      //bar kuuluu nimiavaruuteen foo,
7      //joten f() nakyy taalla
8      void g(){f();}
9    }
10 }
11
12 using namespace foo;
13 void foo::f(){std::cout<<"f()\n";}
14
15 int main(int argc, char **argv) {
16     f();
17     bar::g();
18 }
```

Aliakset

```
1  #include <iostream>
2
3  namespace foo {
4    void f();
5    namespace bar {
6      void g(){f();}
7    }
8  }
9
10 void foo::f(){std::cout<<"f()\n";}
11
12 //Aliakset
13 namespace f=foo;
14 namespace fb=foo::bar;
15
16
17 int main(int argc, char **argv) {
18   f::f();
19   fb::g();
20 }
```

Muita nimiavaruuksien ominaisuuksia

- Globaaliin nimiavaruuteen viitataan notaatiolla `::Funktio()`.
- Anonyymin nimiavaruuden avulla voidaan estää tunnisteiden näkyminen tiedoston ulkopuolelle.

```
1  namespace {
2    //Koodi tanne
3  }
```

- Inline-nimiavaruus (C++11) muistuttaa sisäkkäistä nimiavaruutta, mutta ulompi nimiavaruus näkee myös sisemmän (inline) nimiavaruuden tunnisteet.

```
1  namespace foo {
2    inline namespace inline_bar {...}
3    namespace bar {...}
4  }
```

Smart-pointer

- Tapa toteuttaa automaattinen dynaamisesti varattujen olioiden tuhoaminen.

→ Dynaamisesti luotu olio tuhotaan varmasti tasan yhden kerran.

- Ennen C++11-standardia `auto_ptr` – Vanhentunut (poistuu C++17:ssa?)

→ Älä käytä uusissa ohjelmissa, voi korvata lähes suoraan `unique_ptr`:llä!

- Ei ole olemassa yhtä smart-pointer-tyyppiä joka toimisi oikein kaikissa tilanteissa.
- `unique_ptr` – Kun yksi osoitin osoittaa objektiin.
- `shared_ptr` – Kun useita osoittimia osoittaa objektiin.
- `weak_ptr` – "Viittaus" `shared_ptr`-objektiin.

Motivaatio

```
1  #include <iostream>
2  #include <memory>
3
4  struct Foo {
5      Foo()       { std::cout << "Foo::Foo\n";  }
6      ~Foo()      { std::cout << "Foo::~Foo\n"; }
7      void bar()  { std::cout << "Foo::bar\n";  }
8  };
9
10 int main() {
11   {
12     Foo* p=new Foo;
13     p->bar();
14   }
15   //p tuhottu, mutta oliota ei tuhottu
16
17   {
18     std::unique_ptr<Foo> p_smart(new Foo);
19     p_smart->bar();
20   }
21 }
```

Ajoesimerkki

```
$./a.out
Foo::Foo
Foo::bar
Foo::Foo
Foo::bar
Foo::~Foo
```

unique_ptr – Parametrinvälitys

```
1  #include <iostream>
2  #include <memory>
3
4  struct Foo {
5      Foo()       { std::cout << "Foo::Foo\n";   }
6      ~Foo()      { std::cout << "Foo::~Foo\n";  }
7      void bar()  { std::cout << "Foo::bar\n";   }
8  };
9
10 void Bar(Foo &f) {f.bar();}
11 void FooBar(std::unique_ptr<Foo> &p) {p->bar();}
12 void FooFoo(std::unique_ptr<Foo> p) {p->bar();}
13 void BarFoo(Foo *p) {p->bar();}
14
15 int main() {
16     std::unique_ptr<Foo> p_smart(new Foo);
17     Bar(*p_smart);
18     FooBar(p_smart);
19     FooFoo(p_smart); //Virhe!
20     BarFoo(p_smart); //Virhe!
21 }
```

unique_ptr – säiliöiden käyttö

```
1  #include <iostream>
2  #include <memory>
3  #include <list>
4
5  struct Foo {
6     Foo()      { std::cout << "Foo::Foo\n";  }
7    ~Foo()      { std::cout << "Foo::~Foo\n"; }
8    void bar() { std::cout << "Foo::bar\n";  }
9  };
10
11 int main() {
12   std::list<std::unique_ptr<Foo> > l;
13
14   l.push_back(std::unique_ptr<Foo>(new Foo));
15
16   //Pitaa kayttaa viittausta, kopion teko on
17   //kielletty
18   for(auto &i : l) {
19     i->bar();
20   }
21 }
```

unique_ptr – std::move 1/2

```
1  #include <iostream>
2  #include <memory>
3  #include <list>
4
5  struct Foo {
6    int idx;
7    Foo(int i) {idx=i; std::cout << "Foo::Foo "<<idx<<"\n";}
8    ~Foo()     {std::cout << "Foo::~Foo "<<idx<<"\n";}
9    void bar() {std::cout << "Foo::bar "<<idx<<"\n";}
10 };
11
12 int main() {
13   std::list<std::unique_ptr<Foo> > l;
14
15   l.push_back(std::unique_ptr<Foo>(new Foo(1)));
16   l.push_back(std::unique_ptr<Foo>(new Foo(2)));
```

unique_ptr – std::move 2/2

```
1   {
2     std::unique_ptr<Foo> p=std::move(l.back());
3     p->bar();
4     //Omistus siirrettava takaisin listassa olevalle
5     //osoittimelle.
6     l.back()=std::move(p); // <--
7   }
8   std::cout<<"p tuhottu\n";
9
10  for(auto &i : l) {
11    i->bar();
12  }
13 }
```

Harjoituksia

(Ex 14) Mitä tapahtuu jos edellisessä esimerkissä omistusta ei siirretäkään takaisin listan `l` viimeiselle alkiolle?

Johdanto

- Singleton-tyypillä tarkoitetaan tyyppiä, josta luodaan vain yksi instanssi.
- Tällaiselle tyypille voi olla tarvetta esimerkiksi
  - käyttäjän syötteiden lukemisessa (yksi näppäimistö tms.),
  - peligrafiikan piirrossa (yksi näyttö),
  - lokiin kirjoituksessa (yksi loki),
  - verkkopelin asiakkaassa (yksi palvelin),
  - jne.
- C++ ei suoraan tarjoa singleton määrettä/tyyppiä.
- Yksinkertaisen toteutuksen idea:
  - Estetään muodostimen käyttö ulkopuolisilta (`protected`).
  - Estetään kopioiden teko (kopiomuodostin `=delete` tai private).
  - `static`-tyyppinen osoitin luokan olioon.
  - Oliota käytettään (esimerkiksi) `Instance`-metodin kautta joka huolehtii edellisen osoittimen alustamisesta tarvittaessa.
  - Tätä oliota ei koskaan tuhota!
  - On myös kehittyneempiä tapoja toteuttaa singleton.

Esimerkkitoteutus

```
1  // Declaration
2  class Singleton {
3   public:
4    static Singleton* Instance();
5   protected:
6    Singleton();
7   private:
8    Singleton(const Singleton &)=delete;
9    static Singleton* _instance;
10   ...
11 }
12
13 // Implementation
14 Singleton* Singleton::_instance = NULL;
15
16 Singleton* Singleton::Instance() {
17   if (_instance == NULL) {
18     _instance = new Singleton;
19   }
20   return _instance;
21 }
```

Harjoituksia

(Ex 15) Kirjoita singleton-malli (`template<typename T> class Singleton...`) jonka avulla voit jatkossa toteuttaa singleton-luokkia. Vihje: Muuta `_instance`-muuttujan tyyppi osoittimeksi luokan `T`-olioon, jolloin myös metodi `Instance` palauttaa osoittimen luokan `T` olioon.

Yleistä

- `assert`-makrolla voidaan tarkastaa täyttyykö jokin ennalta määrätty ehto, esimerkiksi onko kutsutun funktion parametrit sallitut.
  - Yleisesti debug-version ohjelmavirheiden jäljittämiseen.
  - Kun virhe havaitaan ohjelma pysähtyy ja virheilmoitus tulostetaan.
  - Esimerkiksi kun funktiolle on välitetty parametrina `NULL`-osoitin.
- Poikkeusten avulla voidaan välittää tieto virhetilanteesta funktion sisältä ulkopuolelle.
  - "Vähemmän vakavien" virhetilanteiden käsittelyyn.
  - Yleensä poikkeus otetaan kiinni ja virhetilanteesta pyritään toipumaan mikäli mahdollista.
  - Esimerkiksi kun tiedostoa ei voida avata.

assert-esimerkki

```
1  #include <iostream>
2  #include <cassert>
3  #include <cstring>
4
5  int StrCmp(char *s1, char *s2) {
6    assert(s1!=NULL);
7    assert(s2!=NULL);
8
9    return strcmp(s1, s2);
10 }
11
12 int main() {
13   char *a=NULL;
14   char *b=NULL;
15
16   a=(char *)"foo";
17
18   std::cout<<StrCmp(a, b)<<"\n";
19 }
```

Ajoesimerkki

- assert aktiivisena

```
$g++ -std=c++11 main.cpp &&  ./a.out
a.out: main.cpp:7: int StrCmp(char*, char*): Assertion
‘s2!=__null’ failed.
Aborted (core dumped)
```

- Jos NDEBUG-makro on määritelty niin assert jätetään huomioimatta.
- Ilman assertia

```
$g++ -DNDEBUG -std=c++11 main.cpp && ./a.out
Segmentation fault (core dumped)
```

Poikkeuskäsittely

- Usein poikkeus-/virhetilanteeseen ei voida reagoida siellä, missä virhe havaitaan.
- Funktion paluuarvoa voidaan käyttää tähän.
  - Lisää ehtorakenteita ja virheen kuljetusta usean kerroksen läpi.
  - Funktion paluuarvoa ei voi käyttää "oikean paluuarvon"palauttamiseen.
    $\Rightarrow$ Koodi muuttuu helposti epäselväksi.
- C++ tarjoaa työkalut välittää virhe ylemmälle tasolle ja käsitellä virhe siellä.

Poikkeuskäsittely

- Poikkeus heitetään `throw`-käskyllä
- Poikkeuskäsittely näyttää seuraavanlaiselta:

```
1
2  try {
3    //Koodi, jossa poikkeus voidaan heittaa
4    //throw-kaskylla
5  } catch(exception_t &e) {
6    //Koodi poikkeuksen exception_t
7    //kasittelemiseksi
8  } catch(anotherexception_t &e) {
9    //Koodi poikkeuksen anotherexception_t
10   //kasittelemiseksi
11 } catch(...) {
12   //Koodi muiden poikkeusten kasittelemiseksi
13 }
```

- Jos sopivaa poikkeuskäsittelijää ei löydy, välittyy poikkeus eteenpäin.
- Poikkeuskäsittelijässä voidaan myös heittää uusi poikkeus.

Yleinen poikkeuskäsittelijä

- Käsittelee kaikki poikkeukset
- `catch(...)`
- Ei ole tietoa siitä, mikä poikkeus on tapahtunut.
- Jos on käytössä, pitää olla viimeisenä.

Poikkeusten määrittely

- Heitettävä poikkeus on jokin tyyppi:
  - `throw myexception_t();` (luokka)
  - `throw myexception_t(__LINE__, __FILE__);` (luokka)
  - `throw "poikkeus";` (`char`)-tyyppi
  - `throw 723;` (`int`)-tyyppi
- Poikkeuksen ei siis tarvitse olla peritty mistään erityisestä poikkeustyypistä.

Esimerkki (1/3)

```
1  #include<string>
2  #include<iostream>
3
4  using namespace std;
5
6  class myexception_t {
7    int line;
8    string file;
9    string error;
10   public:
11   myexception_t(int l, char *f, char *e) {
12     line=l; file=f; error=e;
13   }
14   void Print() {
15     cout<<"Exception:"<<file<<" ("<<line<<") \""<<error<<"↩
           \"\n";
16   }
17 };
```

Esimerkki (2/3)

```
1  class intvector_t {
2    int *p;
3    int n;
4   public:
5    intvector_t(int len) {
6      n=len; p=new int[n];
7    }
8    int &operator[](int i) {
9      if(i<0 || i>=n) {
10       throw myexception_t(__LINE__, __FILE__,
11         "Index out of range");
12     }
13     return p[i];
14   }
15 };
```

Esimerkki (3/3)

```
1  int main(int argc, char **argv) {
2    intvector_t v(10);
3   try {
4    for(int i=0; i<10; i++) {
5      v[i]=i;
6    }
7
8    for(int i=0; i<=10; i++) {
9      cout<<v[i]<<"\n";
10   }
11
12  } catch(myexception_t &e) {
13     e.Print();
14  } catch(...) {
15     cout<<"Unknown exception\n";
16  }
17 }
```

C++–standardissa määriteltyjä poikkeuksia

- Kantaluokkana on `exception`
  - `bad_cast` – `dynamic_cast` epäonnistui
  - `bad_alloc` – `new` epäonnistui
  - `runtime_error`
    - `range_error` – funktion paluuarvoa ei voi esittää paluutyypin avulla
    - `overflow_error` – aritmeettinen ylivuoto
    - `underflow_error` – aritmeettinen ylivuoto

Ohjeita poikkeuskäsittelyn suunnittelemiseen

- Poikkeusvarmuus vaikuttaa luokan suunnitteluun, joten se pitää ottaa huomioon alusta asti.
- Huolehdi muistin ja muiden resurssien vapauttamisesta poikkeustilanteessa.
- Dokumentoi luokan ja funktioiden generoimat poikkeukset huolellisesti. Muista myös ylläpitää dokumenttia!
- Muista koodin uudelleenkäyttö.

Harjoituksia

(Ex 16) Selvitä miten poikkeuksen uudelleenheittäminen (rethrow) toimii. Kirjoita lyhyt esimerkkiohjelma.

(Ex 17) Kuinka määrittelet lähdekoodiin että funktio ei missään tilanteessa heitä poikkeusta?

(Ex 18) Suunnittele oma poikkeusluokka lähtien omista tarpeistasi. Lisää poikkeusten generointi ja niiden käsittely peliprojektiisi. Jatka tätä tarvittaessa myöhemmin.

Historiaa

- Silicon Graphics Inc.:n IRIS GL, vain SGI:n laitteille.
- Sun Microsystems, IBM ja HP kehittivät PHIGS-rajapintaa.

→ SGI:n markkinaosuus laski

  - SGI päätti avata IRIS GL - rajapinnan, mutta lisenssit ja patentit estivät tämän.

- 1992 SGI, MS, IBM ja Intel perustivat OpenGL architectural review board:n (ARB).
- v1.0, Mark Segal ja Kurt Akeley, SGI, 1992-1994
  - `https://www.opengl.org/registry/doc/glspec10.pdf`
- 2006 Khronos Group
- Viimeisin versio 4.5, 2014
- Kurssilla käytetään versiota 3.3
- `https://www.opengl.org/wiki/History_of_OpenGL`
- OpenGL ES 3.2, 2015

Lähteitä

- "The Red Book" – OpenGL Programming Guide, 8th Edition. ISBN 0-321-77303-9, A tutorial and reference book.
  - Sisältää dokumentaation myös varjostimien ohjelmointiin (entinen "The Orange Book")
- `https://www.khronos.org/`
- `https://www.opengl.org/documentation/`
- `https://sgar91.files.wordpress.com/2010/12/opengl-programming-guide-7e.pdf`

OpenGL ES

- OpenGL:n osajoukko sulautettuihin järjestelmiin.
  - Esimerkiksi konsolit, puhelimet ja ajoneuvot.
- Ei sisällä GLU- ja glut-kirjastoja.
- Oma varjostinkieli OpenGL ES SL, ei yhteensopiva GLSL:n kanssa.
- Eri järjestelmät tukevat eri versioita OpenGL ES:stä[1]
  - OpenGL ES 2.0 – Android 2.0, iOS 5
  - OpenGL ES 3.0 – Android 4.3, iOS 7
  - OpenGL ES 3.1 – Android 5.0

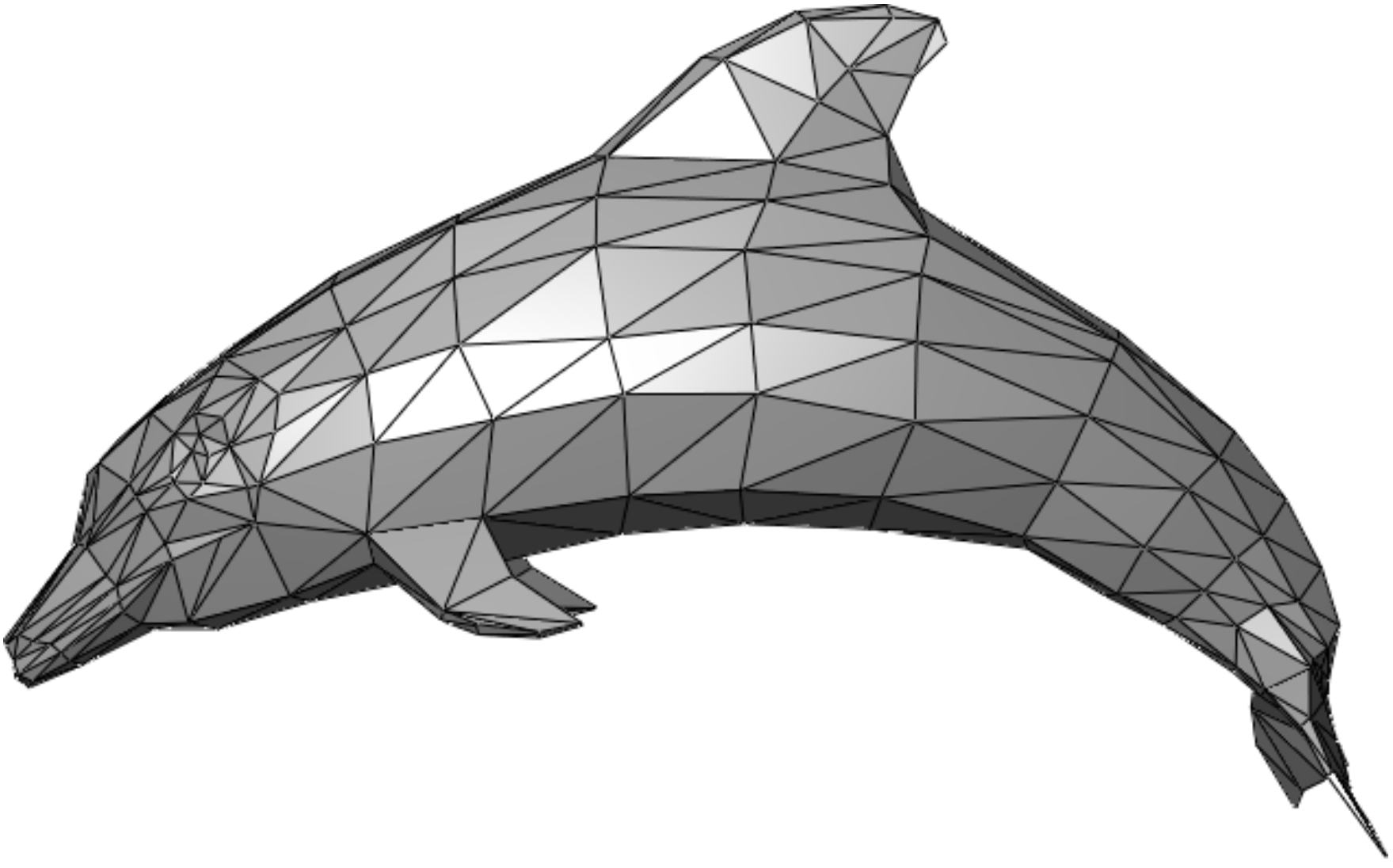
[1] suuntaa-antavasti

Perusteet

Yleensä 3D-kappaleet piirretään siten että:

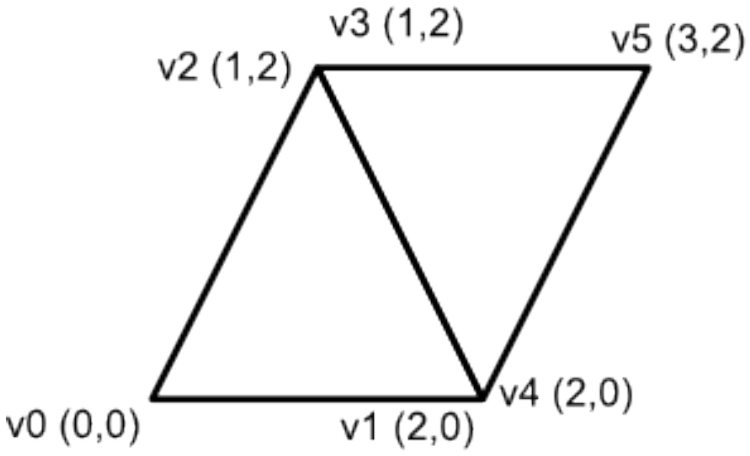
- Vain piirrettävän kappaleen pinta piirretään.
- Piirrettävä pinta jaetaan tasomaisiin monikulmioihin, usein kolmioihin (triangle mesh).
- Yleensä kolmioiden kärkipisteet talletetaan erillisen listaan (vertexlist).
  - Kun useampi kolmio jakaa yhteisen kärkipisteen talletetaan se listaan vain kerran.
  - Kolmioiden kulmapisteiden indeksit talletetaan toiseen listaan (indexlist).
  - On myös mahdollista tallettaa kärkipisteet kolmioittain, jolloin indeksilistaa ei käytetä. (Seuraavat esimerkit on toteutettu näin)
- Kolmioinnin päällä käytetään usein jotain tehosteita kuten esimerkiksi teksturointia, valaistusmalleja jne., joilla pinta saadaan näyttämään luonnollisemmalta.

Kolmiointi



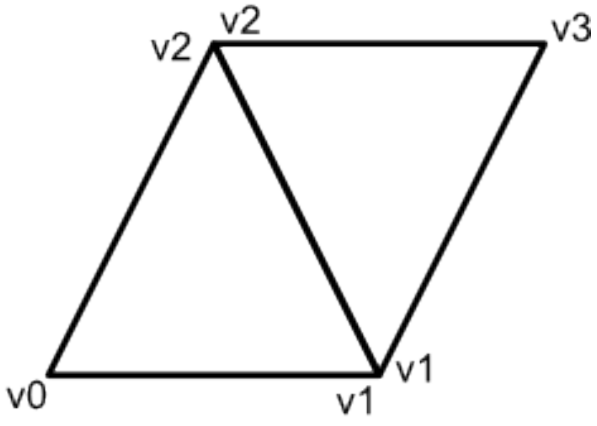
Indeksointi

Without indexing



[0,0, 2,0, 1,2, 1,2, 2,0, 3,2]

With indexing

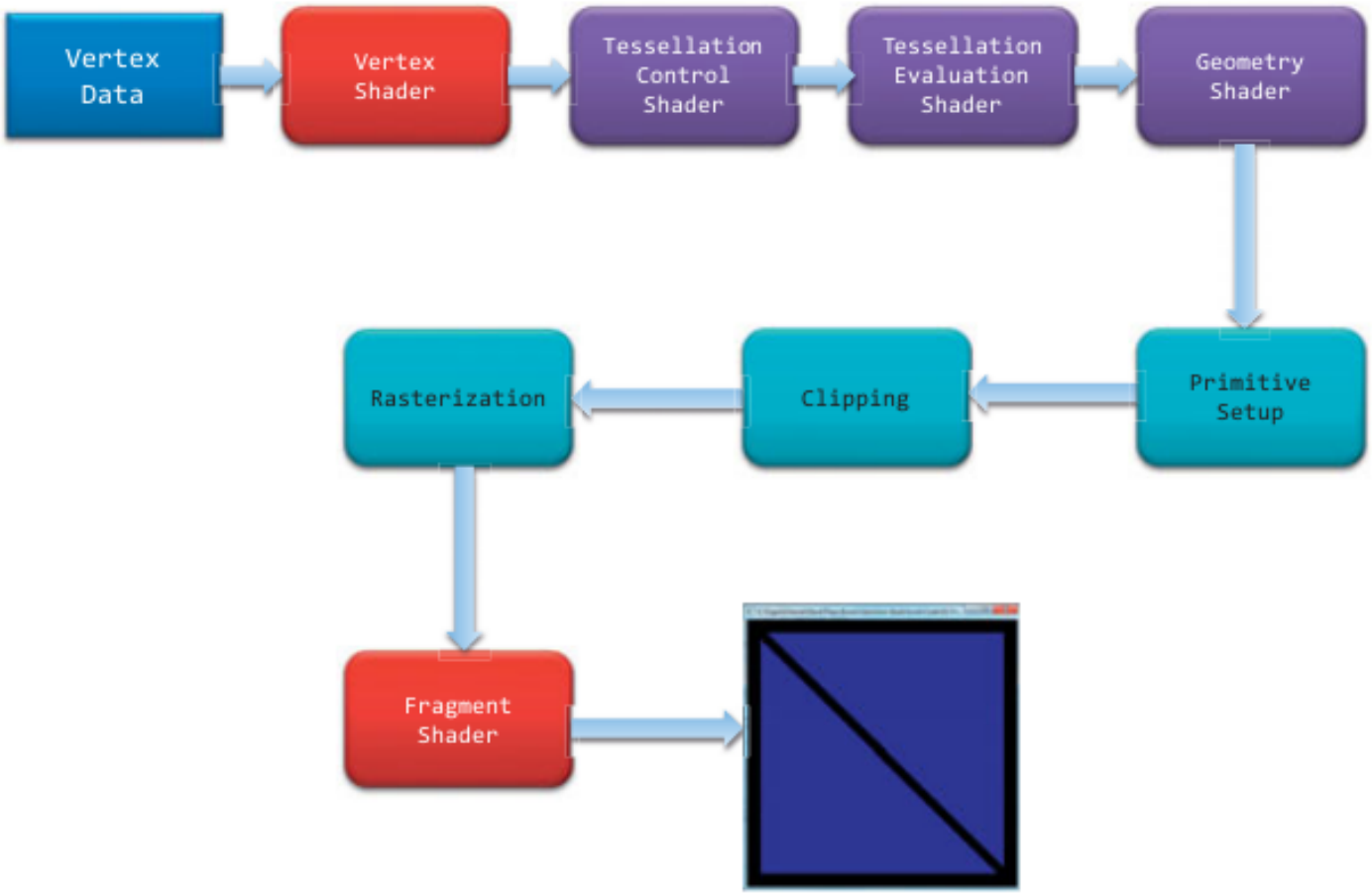


[0,1,2, 2,1,3]

[0,0, 2,0, 1,2, 3,2]

Vertices reused twice

OpenGL-liukuhihna



Esimerkit

- Muokattu `http://www.opengl-tutorial.org/beginners-tutorials/`
- Käyttävät GLFW-kirjastoa, `http://www.glfw.org/`, mutta ovat helposti muokattavissa esimerkiksi glut-rajapinnan päällä toimiviksi.
- Tarvittaessa korvaa `common/shader.cpp`:stä `#include <glew.h>` seuraavilla riveillä:

```
1 #define GL_GLEXT_PROTOTYPES 1
2 #include <GL/glut.h>
```

Kolmio GLFW:llä 1/7

`glfw_kolmio.zip`

```
1  #include <cstdio>
2  #include <cstdlib>
3  #include <GL/glew.h>
4  #include <glfw3.h>
5  #include <glm/glm.hpp>
6  using namespace glm;
7  #include <common/shader.hpp>
8
9  GLFWwindow* window;
10 GLuint programID;
11 GLuint vertexbuffer;
12 GLuint VertexArrayID;
13 GLuint colorbuffer;
```

Kolmio GLFW:llä 2/7

```
1  void Render(void) {
2    glClear( GL_COLOR_BUFFER_BIT );
3    glUseProgram(programID);
4    glEnableVertexAttribArray(0);
5    glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
6    glVertexAttribPointer(
7       0,        // must match the layout in the shader.
8       3,        // size
9       GL_FLOAT,// type
10      GL_FALSE,// normalized
11      0,        // stride
12      (void*)0 // array buffer offset
13      );
```

Kolmio GLFW:llä 3/7

```
1   glEnableVertexAttribArray(1);
2   glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
3   glVertexAttribPointer(
4       1,          // must match the layout in the shader.
5       3,          // size
6       GL_FLOAT,   // type
7       GL_FALSE,   // normalized?
8       0,          // stride
9       (void*)0    // array buffer offset
10      );
11  glDrawArrays(GL_TRIANGLES, 0, 3);
12  glDisableVertexAttribArray(0);
13  glDisableVertexAttribArray(1);
14  glfwSwapBuffers(window);
15 }
```

Kolmio GLFW:llä 4/7

```
1  void Init(void) {
2    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
3
4    glGenVertexArrays(1, &VertexArrayID);
5    glBindVertexArray(VertexArrayID);
6
7    programID = LoadShaders(
8      "VertexShader.vertexshader",
9      "ColorFragmentShader.fragmentshader" );
10
11   static const GLfloat g_vertex_buffer_data[] = {
12     -0.5f, -0.5f, 0.0f, 0.5f,  0.0f, 0.0f,
13     0.0f,  0.5f, 0.0f, };
14
15   glGenBuffers(1, &vertexbuffer);
16   glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
17   glBufferData(GL_ARRAY_BUFFER,
18     sizeof(g_vertex_buffer_data),
19     g_vertex_buffer_data, GL_STATIC_DRAW);
```

Kolmio GLFW:llä 5/7

```
1   static const GLfloat g_color_buffer_data[] = {
2     1.0f,  0.0f, 0.0f,
3     0.0f,  1.0f, 0.0f,
4     0.0f,  0.0f, 1.0f,
5   };
6   glGenBuffers(1, &colorbuffer);
7   glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
8   glBufferData(GL_ARRAY_BUFFER,
9     sizeof(g_color_buffer_data),
10    g_color_buffer_data, GL_STATIC_DRAW);
11 }
12
13 void Uninit(void) {
14   glDeleteBuffers(1, &vertexbuffer);
15   glDeleteBuffers(1, &colorbuffer);
16   glDeleteVertexArrays(1, &VertexArrayID);
17   glDeleteProgram(programID);
18 }
```

Kolmio GLFW:llä 6/7

```
1  int main( void ) {
2    if( !glfwInit() )
3      {
4        fprintf( stderr, "Failed to initialize GLFW\n" );
5        return -1;
6      }
7
8    glfwWindowHint(GLFW_SAMPLES, 4);
9    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
10   glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
11   glfwWindowHint(GLFW_OPENGL_PROFILE,
12                  GLFW_OPENGL_CORE_PROFILE);
13
14   window = glfwCreateWindow( 1024, 768,
15        "Tutorial 02 - Red triangle", NULL, NULL);
16   if( window == NULL ){
17     fprintf( stderr, "Failed to open GLFW window.");
18     glfwTerminate();
19     return -1;
20   }
21   glfwMakeContextCurrent(window);
```

Kolmio GLFW:llä 7/7

```
1   glewExperimental = true; // Needed for core profile
2   if (glewInit() != GLEW_OK) {
3     fprintf(stderr, "Failed to initialize GLEW\n");
4     return -1;
5   }
6
7   glfwSetInputMode(window, GLFW_STICKY_KEYS, GL_TRUE);
8   Init();
9
10  do{
11    Render();
12    glfwPollEvents();
13  }
14  while( glfwGetKey(window, GLFW_KEY_ESCAPE ) != ↩
        GLFW_PRESS &&
15 glfwWindowShouldClose(window) == 0 );
16
17  glfwTerminate();
18
19  return 0;
20 }
```

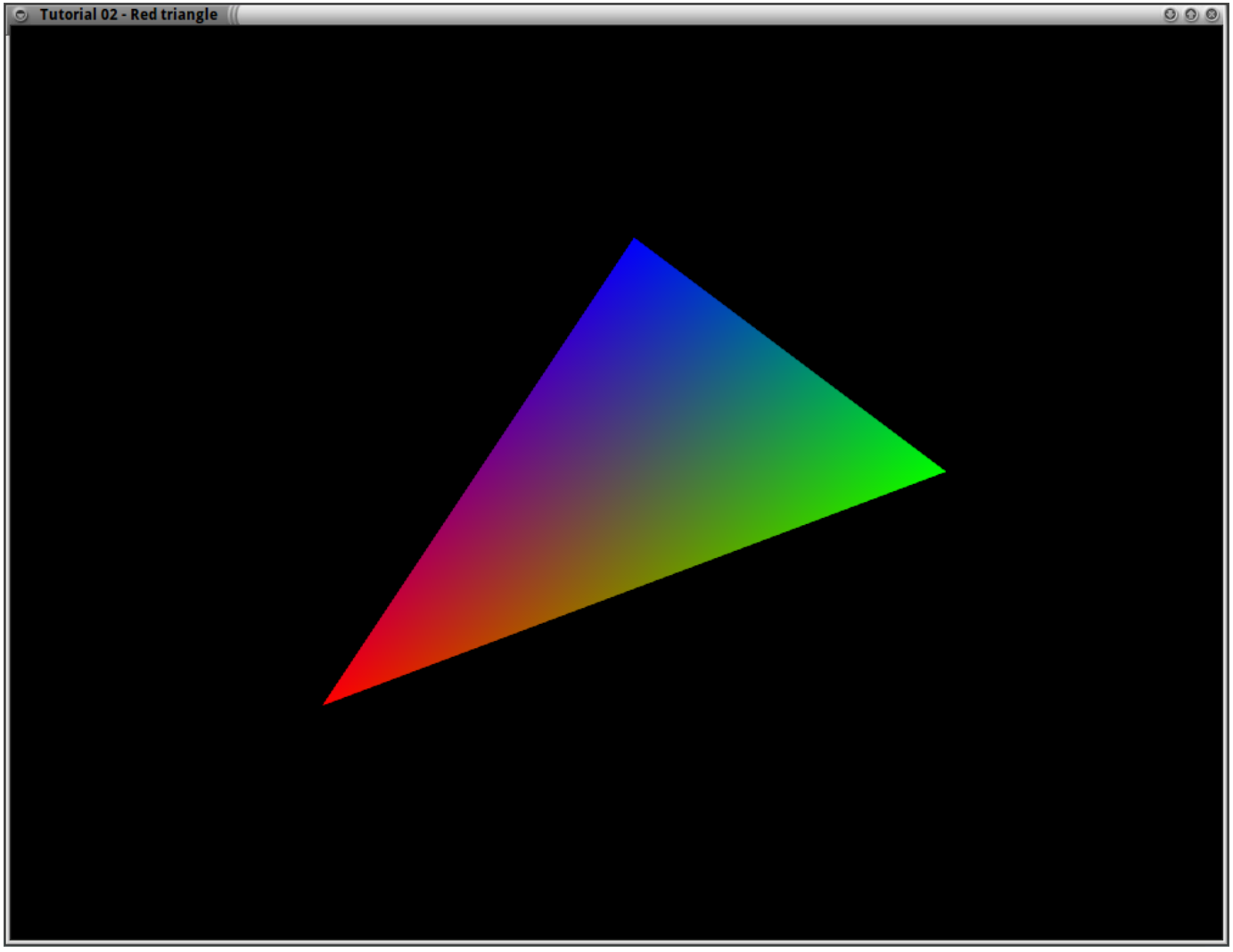
VertexShader.vertexshader

```
#version 330 core

// Input vertex data
layout(location = 0) in vec3 vertexPosition_modelspace;
layout(location = 1) in vec3 vertexColor;

// Output data ; will be interpolated for each fragment.
out vec3 fragmentColor;

/* system outputs
vec4 gl_Position;
float gl_PointSize;
float gl_ClipDistance[]; */

void main(){
gl_Position.xyz =  vertexPosition_modelspace;
gl_Position.w = 1.0; //position

fragmentColor = vertexColor;
}
```

ColorFragmentShader.fragmentshader

```
#version 330 core

// Interpolated values from the vertex shaders
in vec3 fragmentColor;
/*system inputs
in vec4 gl_FragCoord;
in bool gl_FrontFacing;
in vec2 gl_PointCoord; */

// Ouput data
out vec3 color;

void main(){

// Output color = color specified in the vertex shader,
// interpolated between all 3 surrounding vertices
color = fragmentColor;
}
```

Ajoesimerkki



Harjoituksia

(Ex 19) Kirjoita pohja OpenGL-ohjelmalle. Ohjelmiston OpenGL-osat on sijoitettu `renderer.cpp`-tiedostoon, jota pohjan käyttäjä muokkaa. Alla on tiedostoon liittyvä otsikkotiedosto ja seuraavassa diassa itse `renderer.cpp`-tiedoston runko. Voit käyttää GLFW:n sijasta jotain muuta kirjastoa (esim SDL:ää) jos haluat, jolloin `Init`-funktion parametrin tyyppi tietenkin vaihtuu. Voit toteuttaa pohjan myös luokkana jos haluat.

```
1  #ifndef __RENDERER_H__
2  #define __RENDERER_H__
3
4  namespace renderer {
5    void Render(void);
6    void Init(GLFWwindow* w);
7    void Uninit(void);
8  }
9
10 #endif
```

Harjoituksia

(Ex 20) Muokkaa esimerkkiä siten että kolmion värityksestä poistetaan punainen komponentti. Vihje: Värit ilmoitetaan `vec3`-tyyppisellä vektorilla, joka sijoitetaan `color`-muuttujaan fragmentshaderissa. käytä muodostinta `vec3(GLfloat r, GLfloat g, GLfloat b)` ja viittaa vektorin alkioihin `[]`-operaattorilla.

(Ex 21) Muokkaa esimerkkiä (alkuperäistä) siten että kolmion pikselit joiden ruudun koordinaatin $x \times y$ on parillinen maalataan mustalla ja muut kolmion pikselit valkoisella. Vihje: Fragmentshaderin `gl_FragCoord`-parametri.

(Ex 22) Muokkaa esimerkkiä siten että kolmion kärkipisteen x-koordinaatteihin lisätään $0.25 \sin(100.0 * y^2)$. Vihje: vertexshader.

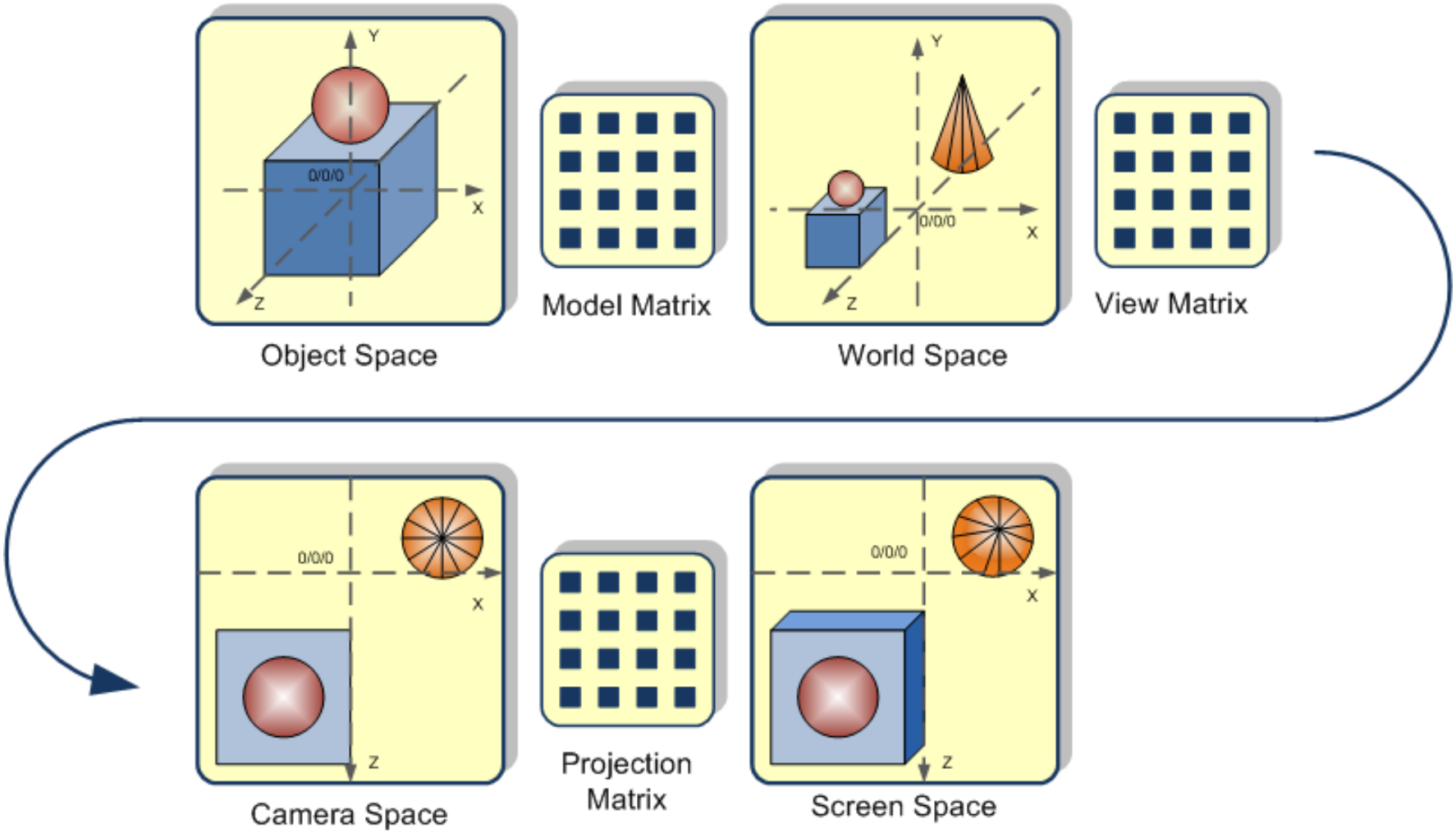
OpenGL Mathematics (GLM)

- `http://glm.g-truc.net/`
- "GLSL-yhteensopiva"
  - Nimeämiset
  - Toiminnallisuus
  - Rakenteet
- Algebralliset operaatiot
- Siirto
- Kierto
- Perspektiivi
- Skaalaus
- Satunnaisluvut (erilaisilla jakaumilla)
- ...

OpenGL:n koordinaatistot

- Objektin koordinaatisto
  - Koordinaatisto jossa graafikko on suunnitellut objektin
- Maailman koordinaatisto
  - Koordinaatisto jossa peli tapahtuu
- Kameran koordinaatisto
  - Maailma suhteessa kameraan
- Kuvaruudin koordinaatisto
  - Kameran kuva projisoituna kuvatasolle (näytölle)

Koordinaatistomuunnokset



Harjoituksia

(Ex 23) Tutustu GLM:ään ja laske sen avulla

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 2 & 0 & 1 & 1 \\ 2 & 0 & 2 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 2 & 0 & 2 & 0 \end{pmatrix} \begin{pmatrix} 3 \\ 4 \\ -2 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \\ 7 \\ 6 \end{pmatrix}$$

.

HUOM: `mat4`:n muodostimessa 16 parametria annetaan sarakkeittain (4 ensimmäistä muodostavat 1. sarakkeen jne).

Esimerkki

Olkoon pisteet $\mathbf{p}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ ja $\mathbf{p}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ määriteltynä objektin koordinaatistossa. Objekti on kiertynyt $\frac{\pi}{2}$ radiaania origon ympäri vastapäivään, skaalattu kertoimella 2 ja sen keskipiste (objektin koordinaatiston origo) sijaitsee maailman koordinaatistossa pisteessä $\begin{pmatrix} 3 \\ -1 \end{pmatrix}$. Laske muunnosmatriisi GLM:n `mat4`-tyyppiä käyttäen. Laske myös pisteiden $\mathbf{p}_1$ ja $\mathbf{p}_2$ sijainti maailman koordinaatistossa.

Ratkaisu 1/4

```
1  #define GLM_FORCE_RADIANS
2  #include <glm/glm.hpp>
3  #include <glm/gtc/matrix_transform.hpp>
4  #include <glm/gtx/transform.hpp>
5  #include <iostream>
6
7  using namespace glm;
8  using namespace std;
9
10 void PrintVec(const vec4 &v) {
11   for(int i=0; i<4; i++) {
12     cout<<v[i]<<"\n";
13   }
14   cout<<"\n";
15 }
```

Ratkaisu 2/4

```
1 void PrintMatrix(const mat4 &m) {
2   for(int i=0;i<4; i++) {
3     for(int j=0; j<4; j++) {
4       cout<<m[j][i]<<" ";
5     }
6     cout<<"\n";
7   }
8   cout<<"\n";
9 }
```

Ratkaisu 3/4

```
1  int main(int argc, char **argv) {
2    vec4 p1(0.0, 0.0, 0.0, 1.0);
3    // kierto -> (0,0) skaalaus -> (0,0) siirto -> (3,-1)
4    // kamera -> (0, -3)
5    // projektio -> (0, -1)
6    vec4 p2(1.0, 0.0, 0.0, 1.0);
7    // kierto -> (0,1) skaalaus -> (0,2) siirto -> (3, 1)
8    // kamera -> (0, -1)
9    // projektio -> (0, -1/3)
10   vec3 x_axis(1.0, 0.0, 0.0);
11   vec3 y_axis(0.0, 1.0, 0.0);
12   vec3 z_axis(0.0, 0.0, 1.0);
```

Ratkaisu 4/4

```
1   vec3 s(2.0); //skaalaus
2   vec3 t(3, -1, 0.0); //siirto
3   vec3 r=z_axis; //kierto
4
5   mat4 R=rotate(3.14159265f/2.0f, r);
6   mat4 S=scale(s);
7   mat4 T=translate(t);
8
9   mat4 T_total=T*S*R;
10  PrintVec(T_total*p1);
11  PrintVec(T_total*p2);
12  PrintMatrix(T_total);
```

Ajoesimerkki

```
3
-1
0
1

3
1
0
1

-8.74228e-08 -2 0 3
2 -8.74228e-08 0 -1
0 0 2 0
0 0 0 1
```

Esimerkki

Asetetaan edelleen kamera (maailman koordinaatiston) pisteeseen $\begin{pmatrix} 3 \\ 2 \end{pmatrix}$ siten että y-akseli osoittaa ylös kameran näkymässä ja x-akseli oikealle. Laske muunnosmatriisi objektin koordinaatistosta kameran koordinaatistoon. Laske myös pisteiden $\mathbf{p}_1$ ja $\mathbf{p}_2$ sijainti kameran koordinaatistossa.

Ratkaisu

```
1     vec3 cam_pos(3, 2, 0);
2     vec3 cam_up=y_axis;
3     vec3 cam_right=x_axis;
4     vec3 cam_front=-z_axis; //oikeakatinen koordinaatisto
5
6     mat4 C=lookAt(cam_pos, cam_pos+cam_front, cam_up);
7     T_total=C*T_total;
8
9     PrintVec(T_total*p1);
10    PrintVec(T_total*p2);
11    PrintMatrix(T_total);
```

Ajoesimerkki

```
-3
0
1

-8.74228e-08
-1
0
1

-8.74228e-08 -2 0 0
2 -8.74228e-08 0 -3
0 0 2 0
0 0 0 1
```

Esimerkki

Kuvaruudun piirtoala on $[-1, 1] \times [-1, 1]$ ja sille halutaan projisoida kameran koordinaatistosta alue $[-4, 4] \times [-3, 3]$. Laske muunnosmatriisi objektin koordinaatistosta ruudun koordinaatistoon. Laske myös pisteiden $\mathbf{p}_1$ ja $\mathbf{p}_2$ sijainti ruudulla. (Näkyvätkö ne ruudulla?)

Ratkaisu

```
1    float v_width=8.0; //viewport in camera coord
2    float v_height=6.0;
3
4    mat4 T_projection=ortho(-0.5f*v_width, 0.5f*v_width, ↩
         -0.5f*v_height, 0.5f*v_height);
5    T_total=T_projection*T_view*T_model;
6    PrintVec(T_total*p1);
7    PrintVec(T_total*p2);
8    PrintMatrix(T_total);
```

Ajoesimerkki

```
-1
0
1

-2.18557e-08
-0.333333
0
1

-2.18557e-08 -0.5 0 0
0.666667 -2.91409e-08 0 -1
0 0 -2 0
0 0 0 1
```

Harjoituksia

(Ex 24) Edellisen esimerkin muunnokset on helppo tarkastaa paperilla ja kynällä. Tarkasta ne! Pisteiden $\mathbf{p}_1$ ja $\mathbf{p}_2$ koordinaatit eri koordinaatistoissa löytyvät aikaisemmasta koodiesimerkistä.

(Ex 25) Olkoon pisteet $\mathbf{p}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $\mathbf{p}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ ja $\mathbf{p}_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ määriteltynä objektin koordinaatistossa. Objekti on kiertynyt $\frac{\pi}{6}$ radiaania origon ympäri myötäpäivään, skaalattu kertoimella 0.3 ja sen keskipiste (objektin koordinaatiston origo) sijaitsee maailman koordinaatistossa pisteessä $\begin{pmatrix} -2 \\ -1 \end{pmatrix}$. Laske muunnosmatriisi GLM:N `mat4`-tyyppiä käyttäen. Laske myös pisteiden $\mathbf{p}_1$ ja $\mathbf{p}_2$ sijainti maailman koordinaatistossa.

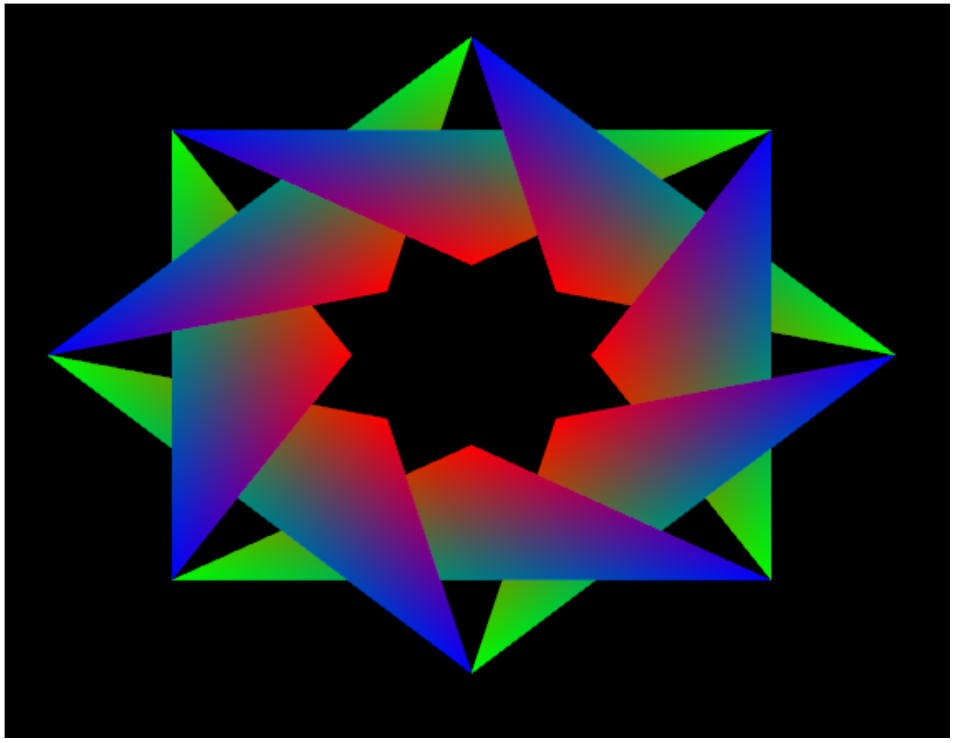
Harjoituksia

(Ex 26) Asetetaan edelleen kamera (maailman koordinaatiston) pisteeseen $\begin{pmatrix} 1.2 \\ 0.1 \end{pmatrix}$ siten että y-akseli osoittaa ylös kameran näkymässä ja x-akseli oikealle. Laske muunnosmatriisi objektin koordinaatistosta kameran koordinaatistoon. Laske myös pisteiden $\mathbf{p}_i$ sijainti kameran koordinaatistossa.

(Ex 27) Kuvaruudun piirtoala on $[-1, 1] \times [-1, 1]$ ja sille halutaan projisoida kameran koordinaatistosta alue $[-3, 3] \times [-3, 3]$. Laske muunnosmatriisi objektin koordinaatistosta ruudun koordinaatistoon. Laske myös pisteiden $\mathbf{p}_i$ sijainti ruudulla. (Näkyvätkö ne ruudulla?)

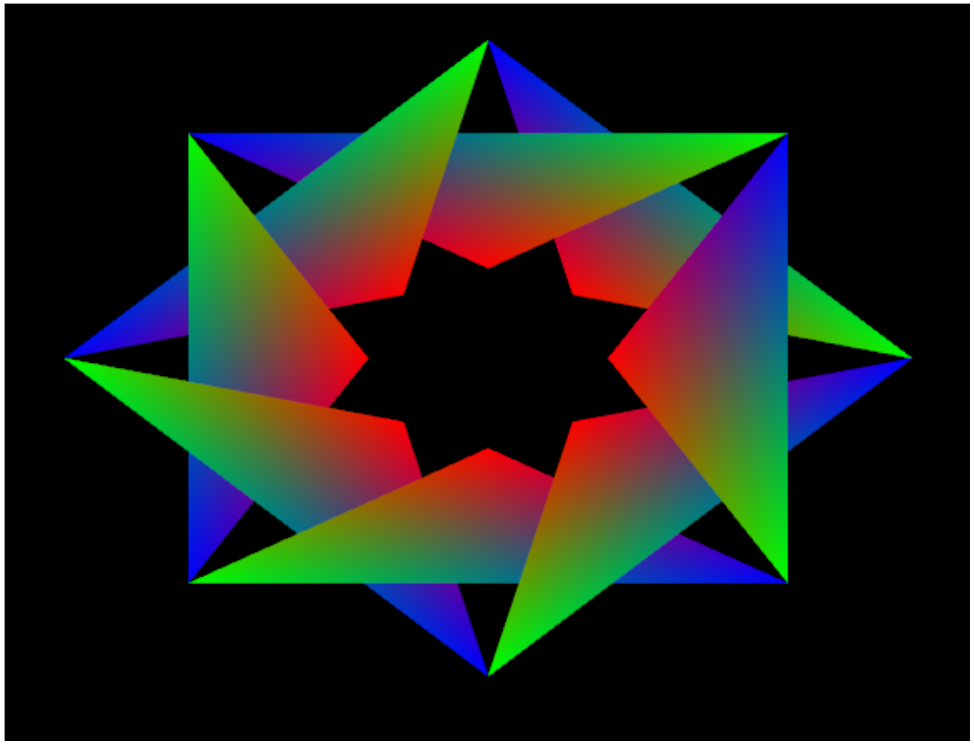
Miksi käytämme z-akselia 2D-grafiikassa? 1/2

- Piirretään kuvio joka koostuu 8 kolmiosta:



Miksi käytämme z-akselia 2D-grafiikassa? 2/2

- Ehkä tulos olikin tällainen:



- OpenGL piirtää objektit siinä järjestyksessä kuin ne ovat ohjelmassa.

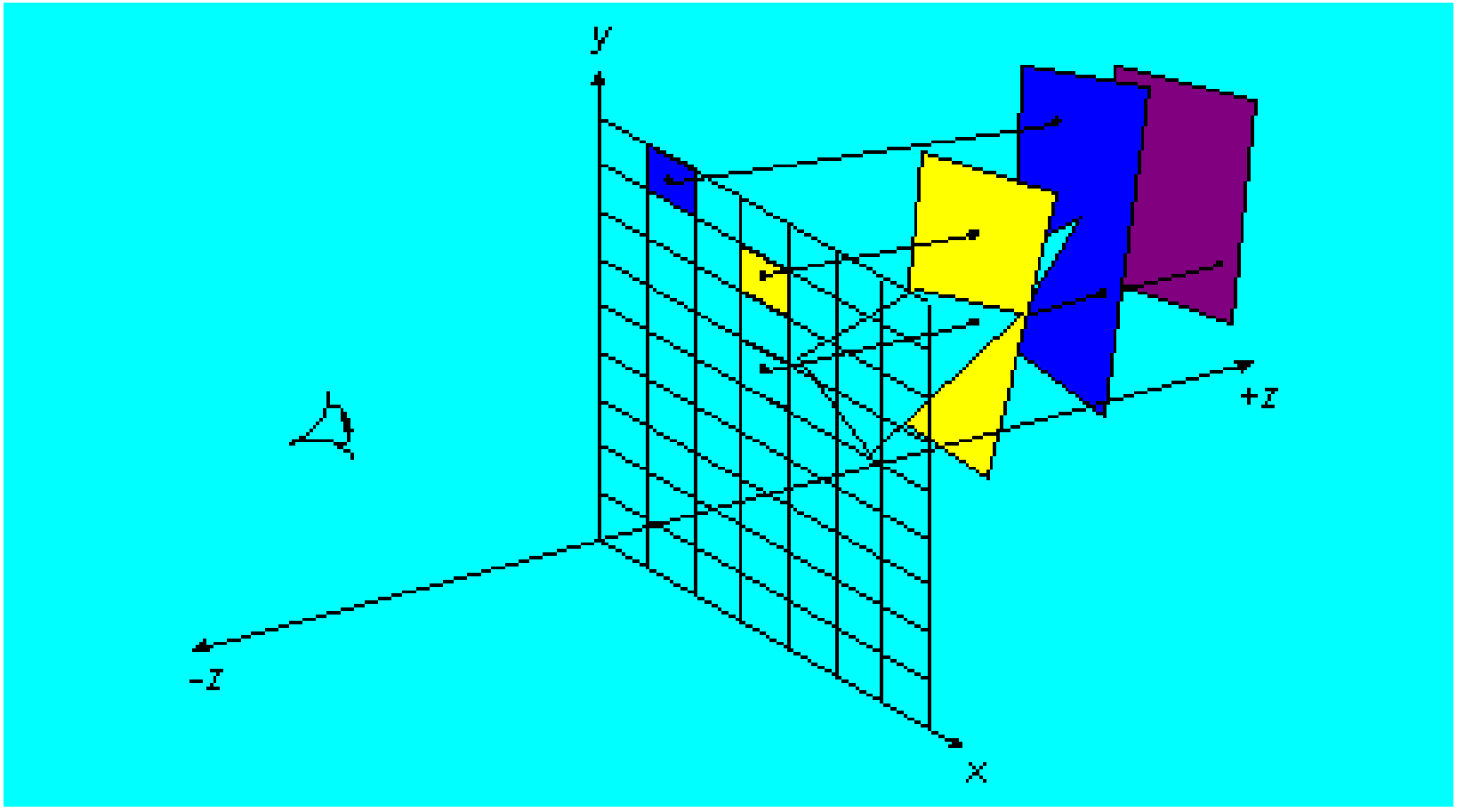
→ Lopputulos ei välttämättä ole halutunlainen.

Z-puskurointi

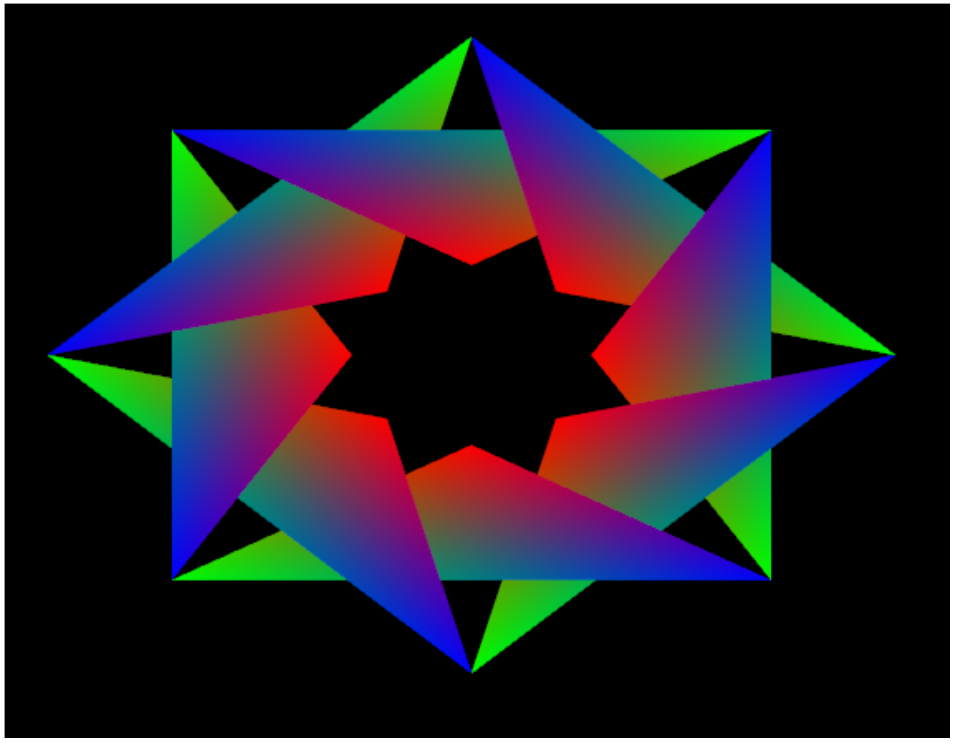
Idea: Jokaiselle pikselille

- Lasketaan pikselille z-koordinaatti.
- Jos pikselin z on suurempi kuin aikaisempi samassa kohdassa ollen pikselin z, eli pikseli on kameran ja aikaisemman pikselin välissä, niin päivitetään z-koordinaatti taulukkoon ja piirretään pikseli.

Z-puskurointi



Kuva jonka halusimme



Koodi kuvaan 1/6

```
1  void renderer::Init(GLFWwindow* w) {
2    window=w;
3    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
4    glEnable(GL_DEPTH_TEST);
5
6    glGenVertexArrays(1, &VertexArrayID);
7    glBindVertexArray(VertexArrayID);
8
9    programID = LoadShaders( "VertexShader.vertexshader",
10                            "FragmentShader.fragmentshader" );
11   MVP_MatrixID = glGetUniformLocation(programID, "MVP");
12
13   static const GLfloat g_vertex_buffer_data[] = {
14     0.1f,  0.1f, -0.1f,
15     0.5f,  0.0f, -0.1f,
16     0.0f,  0.5f, -0.9f,
17   };
```

Koodi kuvaan 2/6

```
1   glGenBuffers(1, &vertexbuffer);
2   glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
3   glBufferData(GL_ARRAY_BUFFER,
4     sizeof(g_vertex_buffer_data),
5     g_vertex_buffer_data, GL_STATIC_DRAW);
6
7   static const GLfloat g_color_buffer_data[] = {
8     1.0f,  0.0f, 0.0f,
9     0.0f,  1.0f, 0.0f,
10    0.0f,  0.0f, 1.0f,
11  };
12
13  N_triangles=1;
14  glGenBuffers(1, &colorbuffer);
15  glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
16  glBufferData(GL_ARRAY_BUFFER,
17    sizeof(g_color_buffer_data),
18    g_color_buffer_data, GL_STATIC_DRAW);
19 }
```

Koodi kuvaan 3/6

```
1  void renderer::Render(void) {
2    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
3    glUseProgram(programID);
4    glm::mat4 MVP_saved=MVP;
5    const int N_repeat=8;
6    for(int j=0; j<N_repeat/N_triangles; j++) {
7      glUniformMatrix4fv(MVP_MatrixID, 1,
8        GL_FALSE, &MVP[0][0]);
9      glEnableVertexAttribArray(0);
10     glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
11     glVertexAttribPointer(
12       VERTEX_POSITION, //layout in the shader.
13       3,        // size
14       GL_FLOAT,// type
15       GL_FALSE,// normalized
16       0,        // stride
17       (void*)0 // array buffer offset
18     );
```

Koodi kuvaan 4/6

```
1  glEnableVertexAttribArray(1);
2      glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
3      glVertexAttribPointer(
4        VERTEX_COLOR,        // must match the layout in the ↩
             shader.
5        3,          // size
6        GL_FLOAT,// type
7        GL_FALSE,// normalized?
8        0,          // stride
9        (void*)0 // array buffer offset
10       );
11     glDrawArrays(GL_TRIANGLES, 0, 3*N_triangles);
12     MVP=glm::rotate(2.0f*3.14159265f
13         /(N_repeat/N_triangles),
14       glm::vec3(0.0f, 0.0f, 1.0f))*MVP;
15   }
```

Koodi kuvaan 5/6

```
1   MVP=MVP_saved;
2   glDisableVertexAttribArray(0);
3   glDisableVertexAttribArray(1);
4   glfwSwapBuffers(window);
5 }
```

Koodi kuvaan 6/6

```
1 void renderer::Uninit(void) {
2   glDeleteBuffers(1, &vertexbuffer);
3   glDeleteBuffers(1, &colorbuffer);
4   glDeleteVertexArrays(1, &VertexArrayID);
5   glDeleteProgram(programID);
6 }
```

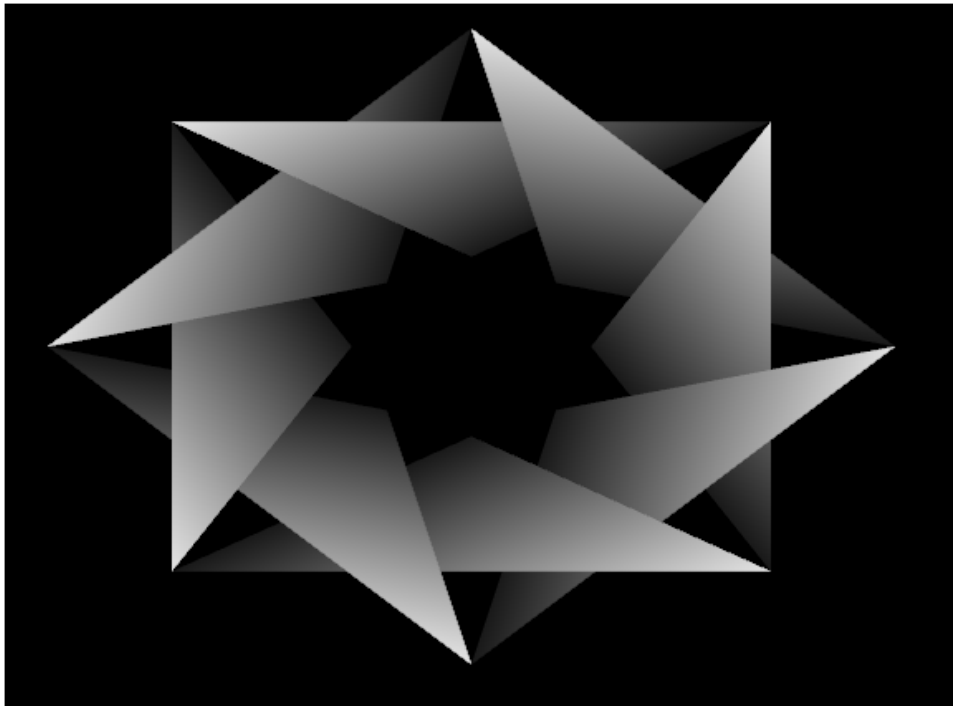
VertexShader.vertexshader

```
1  #version 330 core
2
3  //Defined also in vertexshader.h
4  #define VERTEX_POSITION 0
5  #define VERTEX_COLOR 1
6
7  layout(location = VERTEX_POSITION) in vec3 ↩
       vertexPosition_modelspace;
8  layout(location = VERTEX_COLOR) in vec3 vertexColor;
9
10 out vec3 fragmentColor;
11 /* system outputs
12 vec4 gl_Position;
13 float gl_PointSize;
14 float gl_ClipDistance[]; */
15 uniform mat4 MVP;
16
17 void main() {
18 gl_Position =  MVP * vec4(vertexPosition_modelspace,1);
19         fragmentColor = vertexColor;
20 }
```

FragmentShader.fragmentshader

```
1  #version 330 core
2
3  in vec3 fragmentColor;
4  /*system inputs
5  in vec4 gl_FragCoord;
6  in bool gl_FrontFacing;
7  in vec2 gl_PointCoord; */
8  out vec3 color; // Output data
9
10 void main(){
11   color = fragmentColor;
12 }
```

Z-arvot

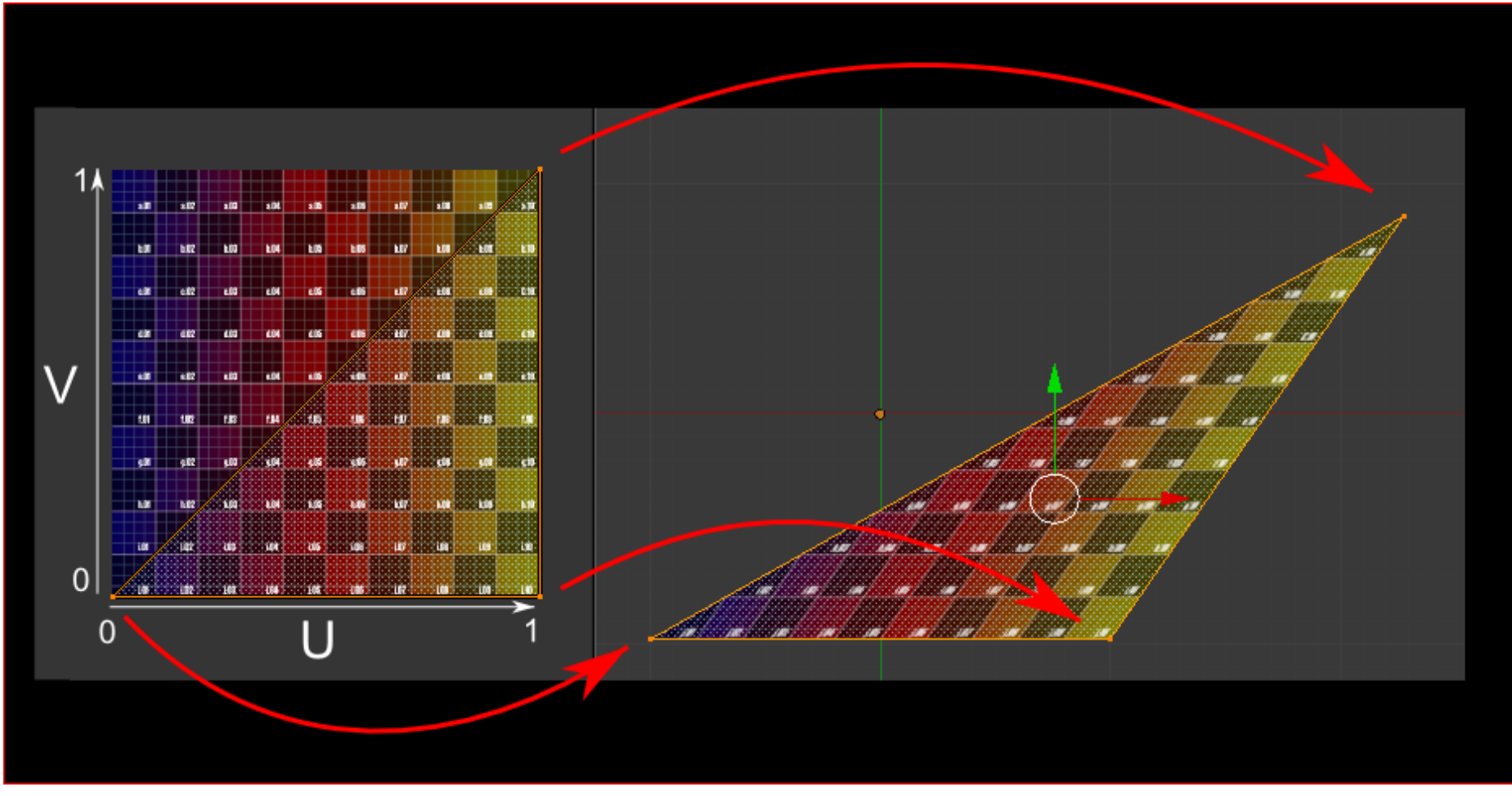


Harjoituksia

(Ex 28) Muokkaa edellistä esimerkkiä siten että kuvio pyörii origon ympäri.

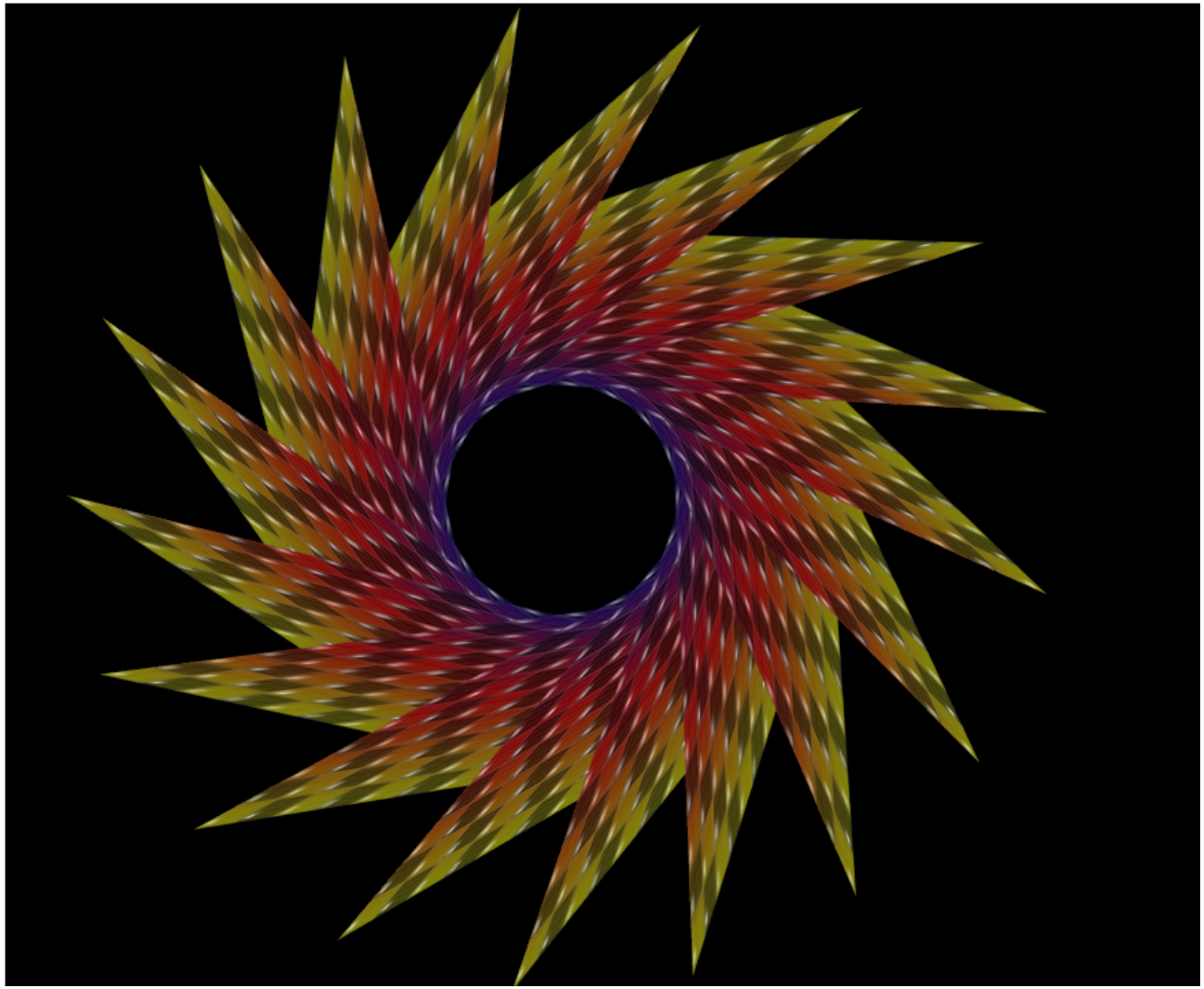
(Ex 29) Muokkaa edellistä edelleen s.e kukin kolmio pyörii objektin koordinaatiston origon ympäri ja on skaalattu kertoimella 1.2.

Texturemapping



Texturemapping

Esimerkki Texturoinnista, indeksoinnista, sekä uniform-parametreista shadereille.



Esimerkki texturoinnista ja indeksoinnista 1/12

```
1  #define GLM_FORCE_RADIANS
2  #include <cstdio>
3  #include <cstdlib>
4  #include <GL/glew.h>
5  #include <glfw3.h>
6  #include <glm/glm.hpp>
7  #include <glm/gtc/matrix_transform.hpp>
8  #include <glm/gtx/transform.hpp>
9  #include <common/shader.hpp>
10 #include <common/texture.hpp>
11 #include <iostream>
12
13 #include "renderer.h"
14 #include "vertexshader.h"
```

Esimerkki texturoinnista ja indeksoinnista 2/12

```
1  namespace {
2    GLuint programID;
3    GLuint vertexbuffer;
4    GLuint VertexArrayID;
5    GLuint indexbuffer;
6    GLFWwindow* window;
7    float alpha=0.0f;
8    glm::mat4 MVP(1.0);
9    GLuint MVP_MatrixID;
10   GLuint TextureID;
11   GLuint uvbuffer;
12   GLuint Texture;
13 };
14
15 void renderer::FramebufferSizeCallback(
16   GLFWwindow* window, int width, int height) {
17   glViewport(0, 0, width, height);
18 }
19 // main.cpp:ssa: glfwSetFramebufferSizeCallback(window, ↩
       renderer::FramebufferSizeCallback);
```

Esimerkki texturoinnista ja indeksoinnista 3/12

```
1  void renderer::Render(void) {
2    const int N_repeat=17;
3
4    glm::vec3 x_axis(1.0, 0.0, 0.0);
5    glm::vec3 y_axis(0.0, 1.0, 0.0);
6    glm::vec3 z_axis(0.0, 0.0, 1.0);
7
8    glm::vec3 cam_pos(0, 0, 0);
9    glm::vec3 cam_up=y_axis;
10   glm::vec3 cam_right=x_axis;
11   glm::vec3 cam_front=-z_axis; //oikeakatinen ↩
         koordinaatisto
12   glm::mat4 P=glm::lookAt(cam_pos, cam_pos+cam_front,
13     cam_up);
```

Esimerkki texturoinnista ja indeksoinnista 4/12

```
1  glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
2  glUseProgram(programID);
3
4  glActiveTexture(GL_TEXTURE0);
5  glBindTexture(GL_TEXTURE_2D, Texture);
6  // Use Texture Unit 0
7  glUniform1i(TextureID, 0);
8
9  int width, height;
10 glfwGetFramebufferSize(window, &width, &height);
11 glm::mat4 V=glm::ortho(-1.0f, 1.0f,
12    -1.0f*height/width, 1.0f*height/width);
13 glm::mat4 VP=V*P;
```

Esimerkki texturoinnista ja indeksoinnista 5/12

```
1  for(int j=0; j<N_repeat; j++) {
2    glm::mat4 M=
3      glm::rotate(alpha+j*2.0f*3.14159265f/N_repeat,
4        z_axis)*
5      glm::translate(glm::vec3(0.0, 0.3, 0))*
6      glm::scale(glm::vec3(0.75));
7    MVP=VP*M;
```

Esimerkki texturoinnista ja indeksoinnista 6/12

```
1   glUniformMatrix4fv(MVP_MatrixID, 1, GL_FALSE, &MVP↩
        [0][0]);
2   glEnableVertexAttribArray(0);
3   glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
4   glVertexAttribPointer(
5     VERTEX_POSITION,
6     3,        // size
7     GL_FLOAT,// type
8     GL_FALSE,// normalized
9     0,        // stride
10    (void*)0 // array buffer offset
11    );
```

Esimerkki texturoinnista ja indeksoinnista 7/12

```
1     glEnableVertexAttribArray(1);
2     glBindBuffer(GL_ARRAY_BUFFER, uvbuffer);
3     glVertexAttribPointer(
4       TEXTURE_DATA,
5       2,  // size : U+V => 2
6       GL_FLOAT,  // type
7       GL_FALSE,  // normalized?
8       0,         // stride
9       (void*)0   // array buffer offset
10      );
11    glDrawElements(GL_TRIANGLES, 3,
12      GL_UNSIGNED_BYTE, (GLvoid*)0);
13  }
```

Esimerkki texturoinnista ja indeksoinnista 8/12

```
1    alpha+=0.005f;
2    glDisableVertexAttribArray(0);
3    glDisableVertexAttribArray(1);
4    glfwSwapBuffers(window);
5  }
```

Esimerkki texturoinnista ja indeksoinnista 9/12

```
1  void renderer::Init(GLFWwindow* w) {
2    window=w;
3    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
4    glEnable(GL_DEPTH_TEST);
5
6    glGenVertexArrays(1, &VertexArrayID);
7    glBindVertexArray(VertexArrayID);
8    programID = LoadShaders( "VertexShader.vertexshader",
9      "FragmentShader.fragmentshader" );
10   MVP_MatrixID = glGetUniformLocation(programID, "MVP");
11
12   static const GLfloat g_vertex_buffer_data[] = {
13     -0.1f,  -0.2f, -0.85f,
14     0.85f,  0.0f, -0.2f,
15     0.1f,  0.1f, -0.9f,
16   };
17   glGenBuffers(1, &vertexbuffer);
18   glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
19   glBufferData(GL_ARRAY_BUFFER,
20     sizeof(g_vertex_buffer_data),
21     g_vertex_buffer_data, GL_STATIC_DRAW);
```

Esimerkki texturoinnista ja indeksoinnista 10/12

```
1  static const GLubyte g_indices[] ={
2    0, 1, 2,
3  };
4  glGenBuffers(1, &indexbuffer);
5  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexbuffer);
6  glBufferData(GL_ELEMENT_ARRAY_BUFFER,
7    sizeof(g_indices), g_indices, GL_STATIC_DRAW);
```

Esimerkki texturoinnista ja indeksoinnista 11/12

```
1   static const GLfloat g_uv_buffer_data[] = {
2     0.0, 0.0,
3     1.0, 0.0,
4     1.0, 1.0,
5   };
6   glGenBuffers(1, &uvbuffer);
7   glBindBuffer(GL_ARRAY_BUFFER, uvbuffer);
8   glBufferData(GL_ARRAY_BUFFER, sizeof(g_uv_buffer_data),
9     g_uv_buffer_data, GL_STATIC_DRAW);
10  TextureID  = glGetUniformLocation(programID, "↩
        myTextureSampler");
11
12  Texture = loadBMP_custom("./uvtemplate.bmp");
13 }
```

Esimerkki texturoinnista ja indeksoinnista 12/12

```
1 void renderer::Uninit(void) {
2   glDeleteBuffers(1, &vertexbuffer);
3   glDeleteBuffers(1, &uvbuffer);
4   glDeleteVertexArrays(1, &VertexArrayID);
5   glDeleteProgram(programID);
6 }
```

vertexshader.h

```
1 #ifndef __VERTEXSHADER_H__
2 #define __VERTEXSHADER_H__
3
4 //defined also in vertexshader
5 #define VERTEX_POSITION 0
6 #define TEXTURE_DATA 1
7
8 #endif
```

VertexShader.vertexshader

```
1  #version 330 core
2
3  //Defined also in vertexshader.h
4  #define VERTEX_POSITION 0
5  #define TEXTURE_DATA 1
6
7  layout(location = VERTEX_POSITION)
8    in vec3 vertexPosition_modelspace;
9  layout(location = TEXTURE_DATA) in vec2 vertexUV;
10 uniform mat4 MVP;
11
12 out vec2 UV;
13 /* system outputs
14 vec4 gl_Position;
15 float gl_PointSize;
16 float gl_ClipDistance[];*/
17
18 void main() {
19   gl_Position =  MVP * vec4(vertexPosition_modelspace,1);
20   UV = vertexUV;
21 }
```

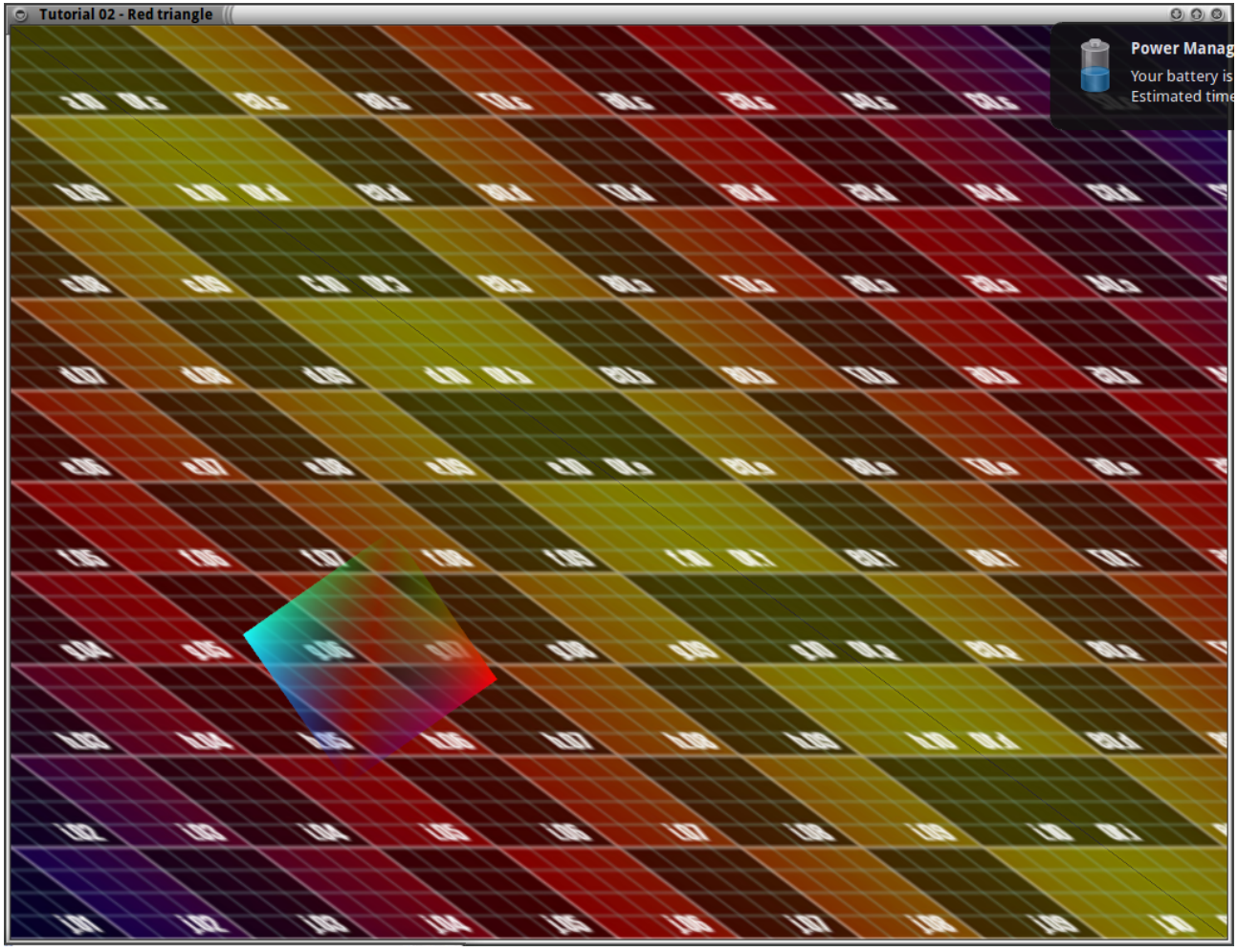
Fragmentshader.fragmentshader

```
1  #version 330 core
2
3  in vec2 UV;
4  uniform sampler2D myTextureSampler; // Texture data
5  /*system inputs
6  in vec4 gl_FragCoord;
7  in bool gl_FrontFacing;
8  in vec2 gl_PointCoord; */
9  out vec3 color; // Output data
10
11 void main(){
12   color=texture(myTextureSampler, UV).rgb;
13 }
```

Harjoituksia

(Ex 30) Kirjoita ohjelma joka piirtää näytölle kaksi teksturoitua nelikulmiota jotka liikkuvat kahdeksikon muotoista rataa.

Esimerkki


Tutorial 02 - Red triangle
Power Manag
Your battery is
Estimated time

Läpinäkyvyys 1/18

```
1  void renderer::Render(void) {
2    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
3
4    DrawBackground();
5    DrawBox();
6
7    glfwSwapBuffers(window);
8  }
9
10 void renderer::Init(GLFWwindow* w) {
11   window=w;
12   glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
13
14   InitBackground();
15   InitBox();
16
17   glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
18 }
```

Läpinäkyvyys 2/18

```
1  void InitBackground() {
2    glGenVertexArrays(1, &textureVertexArrayID);
3    glBindVertexArray(textureVertexArrayID);
4
5    textureProgramID = LoadShaders( "Texture.VertexShader",
6      "Texture.FragmentShader");
7
8   static const GLfloat texture_vertex_buffer_data[] = {
9      -1.0, -1.0,  0.9,
10      1.0, -1.0,  0.9,
11     -1.0, 1.0,  0.9,
12      1.0,  1.0,  0.9,
13   };
14
15   glGenBuffers(1, &texturevertexbuffer);
16   glBindBuffer(GL_ARRAY_BUFFER, texturevertexbuffer);
17   glBufferData(GL_ARRAY_BUFFER,
18     sizeof(texture_vertex_buffer_data),
19     texture_vertex_buffer_data, GL_STATIC_DRAW);
```

Läpinäkyvyys 3/18

```
1  static const GLubyte texture_indices[] ={
2    0, 1, 2, 1 ,3, 2
3  };
4
5  glGenBuffers(1, &textureindexbuffer);
6  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,
7    textureindexbuffer);
8  glBufferData(GL_ELEMENT_ARRAY_BUFFER,
9    sizeof(texture_indices), texture_indices,
10   GL_STATIC_DRAW);
```

Läpinäkyvyys 4/18

```
1  static const GLubyte texture_indices[] ={
2    0, 1, 2, 1 ,3, 2
3  };
4
5  glGenBuffers(1, &textureindexbuffer);
6  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,
7    textureindexbuffer);
8  glBufferData(GL_ELEMENT_ARRAY_BUFFER,
9    sizeof(texture_indices),
10   texture_indices, GL_STATIC_DRAW);
```

Läpinäkyvyys 5/18

```
1   static const GLfloat g_uv_buffer_data[] = {
2     0.0, 0.0,
3     1.0, 0.0,
4     1.0, 1.0,
5     0.0, 1.0,
6   };
7   glGenBuffers(1, &uvbuffer);
8   glBindBuffer(GL_ARRAY_BUFFER, uvbuffer);
9   glBufferData(GL_ARRAY_BUFFER, sizeof(g_uv_buffer_data),
10    g_uv_buffer_data, GL_STATIC_DRAW);
11
12  TextureID  = glGetUniformLocation(textureProgramID,
13    "myTextureSampler");
14
15  Texture = loadBMP_custom("./uvtemplate.bmp");
16
17 }
```

Läpinäkyvyys 6/18

```
1 void InitBox() {
2   glGenVertexArrays(1, &VertexArrayID);
3   glBindVertexArray(VertexArrayID);
4
5   programID = LoadShaders( "VertexShader.vertexshader",
6     "FragmentShader.fragmentshader");
7   MVP_MatrixID = glGetUniformLocation(programID, "MVP");
8   wh_VectorID = glGetUniformLocation(programID, "wh");
```

Läpinäkyvyys 7/18

```
1  static const GLfloat g_vertex_buffer_data[] = {
2    -0.1f,  -0.1f, -0.0f,
3    0.1f,  -0.1f, -0.0f,
4    -0.1f,  0.1f, -0.0f,
5    0.1f,  0.1f, -0.0f,
6  };
7
8  glGenBuffers(1, &vertexbuffer);
9  glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
10 glBufferData(GL_ARRAY_BUFFER,
11   sizeof(g_vertex_buffer_data),
12   g_vertex_buffer_data, GL_STATIC_DRAW);
```

Läpinäkyvyys 8/18

```
1  static const GLfloat g_color_buffer_data[] = {
2    1.0f,  0.0f, 0.0f, 1.0f,
3    0.0f,  1.0f, 0.0f, 0.0f,
4    0.0f,  0.0f, 1.0f, 0.0f,
5    0.1f,  1.0f, 1.0f, 1.0f,
6  };
7
8  glGenBuffers(1, &colorbuffer);
9  glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
10 glBufferData(GL_ARRAY_BUFFER,
11   sizeof(g_color_buffer_data),
12   g_color_buffer_data, GL_STATIC_DRAW);
```

Läpinäkyvyys 9/18

```
1   static const GLubyte g_indices[] ={
2     0, 1, 2, 1 ,3, 2
3   };
4   glGenBuffers(1, &indexbuffer);
5   glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexbuffer);
6   glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(g_indices),
7     g_indices, GL_STATIC_DRAW);
8
9   N_vertex=sizeof(g_indices)/sizeof(*g_indices);
10 }
```

Läpinäkyvyys 10/18

```
1 void DrawBackground() {
2   glDisable(GL_BLEND);
3   glUseProgram(textureProgramID);
4
5   glActiveTexture(GL_TEXTURE0);
6   glBindTexture(GL_TEXTURE_2D, Texture);
7   glUniform1i(TextureID, 0);
```

Läpinäkyvyys 11/18

```
1  glEnableVertexAttribArray(0);
2  glBindBuffer(GL_ARRAY_BUFFER, texturevertexbuffer);
3  glVertexAttribPointer(
4    VERTEX_POSITION,
5    3,        // size
6    GL_FLOAT,// type
7    GL_FALSE,// normalized
8    0,        // stride
9    (void*)0 // array buffer offset
10 );
```

Läpinäkyvyys 12/18

```
1   glEnableVertexAttribArray(1);
2   glBindBuffer(GL_ARRAY_BUFFER, uvbuffer);
3   glVertexAttribPointer(
4     TEXTURE_DATA,
5     2,          // size : U+V => 2
6     GL_FLOAT,   // type
7     GL_FALSE,   // normalized?
8     0,          // stride
9     (void*)0    // array buffer offset
10  );
11    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE,
12      (GLvoid*)0);
13  glDisableVertexAttribArray(0);
14  glDisableVertexAttribArray(1);
15 }
```

Läpinäkyvyys 13/18

```
1  void DrawBox() {
2    glEnable(GL_BLEND);
3    glm::vec3 x_axis(1.0, 0.0, 0.0);
4    glm::vec3 y_axis(0.0, 1.0, 0.0);
5    glm::vec3 z_axis(0.0, 0.0, 1.0);
6
7    glm::vec3 cam_pos(0, 0, 0);
8    glm::vec3 cam_up=y_axis;
9    glm::vec3 cam_right=x_axis;
10   glm::vec3 cam_front=-z_axis;
11   glm::mat4 P=glm::lookAt(cam_pos, cam_pos+cam_front,
12     cam_up);
```

Läpinäkyvyys 14/18

```
1  glUseProgram(programID);
2  int width, height;
3  glfwGetFramebufferSize(window, &width, &height);
4  glm::mat4 V=glm::ortho(-1.0f, 1.0f,
5    -1.0f*height/width, 1.0f*height/width);
6  glm::mat4 VP=V*P;
7  glm::mat4 M=
8    glm::rotate(alpha, z_axis)*
9    glm::translate(glm::vec3(0.0, 0.5, 0))*
10   glm::scale(glm::vec3(1.5));
11 MVP=VP*M;
12
13 glUniformMatrix4fv(MVP_MatrixID, 1,
14   GL_FALSE, &MVP[0][0]);
15 glUniform2fv(wh_VectorID, 1, &wh[0]);
```

Läpinäkyvyys 15/18

```
1  glEnableVertexAttribArray(0);
2  glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
3  glVertexAttribPointer(
4    VERTEX_POSITION, //layout in the shader.
5    3,        // size
6    GL_FLOAT,// type
7    GL_FALSE,// normalized
8    0,        // stride
9    (void*)0 // array buffer offset
10   );
```

Läpinäkyvyys 16/18

```
1   glEnableVertexAttribArray(1);
2   glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
3   glVertexAttribPointer(
4     VERTEX_COLOR,
5     4,          // size
6     GL_FLOAT,// type
7     GL_FALSE,// normalized?
8     0,          // stride
9     (void*)0 // array buffer offset
10    );
11  glDrawElements(GL_TRIANGLES, N_vertex,
12    GL_UNSIGNED_BYTE, (GLvoid*)0);
13
14  alpha+=0.005f;
15  glDisableVertexAttribArray(0);
16  glDisableVertexAttribArray(1);
17 }
```

Läpinäkyvyys 17/18

```
1  void renderer::Uninit(void) {
2    glDeleteBuffers(1, &vertexbuffer);
3    glDeleteBuffers(1, &colorbuffer);
4    glDeleteBuffers(1, &texturevertexbuffer);
5    glDeleteBuffers(1, &textureindexbuffer);
6    glDeleteVertexArrays(1, &VertexArrayID);
7    glDeleteVertexArrays(1, &textureVertexArrayID);
8    glDeleteProgram(programID);
9    glDeleteProgram(textureProgramID);
10 }
```

Läpinäkyvyys 18/18

```
1  namespace {
2    GLuint programID;
3    GLuint vertexbuffer;
4    GLuint VertexArrayID;
5    GLuint colorbuffer;
6    GLuint indexbuffer;
7    GLFWwindow* window;
8    float alpha=0.0f;
9    glm::mat4 MVP(1.0);
10   GLuint MVP_MatrixID;
11   glm::vec2 wh;
12   GLuint wh_VectorID;
13   GLuint N_vertex;
14   GLuint texturevertexbuffer;
15   GLuint textureVertexArrayID;
16   GLuint textureindexbuffer;
17   GLuint uvbuffer;
18   GLuint TextureID;
19   GLuint Texture;
20   GLuint textureProgramID;
21 };
```

vertexshader.h

```
1 #ifndef __VERTEXSHADER_H__
2 #define __VERTEXSHADER_H__
3
4 //defined also in vertexshader
5 #define VERTEX_POSITION 0
6 #define VERTEX_COLOR 1
7 #define TEXTURE_DATA 1
8 #endif
```

VertexShader.vertexshader

```
1  #version 330 core
2
3  #define VERTEX_POSITION 0
4  #define VERTEX_COLOR 1
5
6  layout(location = VERTEX_POSITION) in
7    vec3 vertexPosition_modelspace;
8  layout(location = VERTEX_COLOR) in vec4 vertexColor;
9
10 out vec4 fragmentColor;
11 uniform mat4 MVP;
12
13 void main() {
14 gl_Position =  MVP * vec4(vertexPosition_modelspace,1);
15         fragmentColor = vertexColor;
16 }
```

FragmentShader.fragmentshader

```
1  #version 330 core
2
3  in vec4 fragmentColor;
4  uniform vec2 wh;
5  /*system inputs
6  in vec4 gl_FragCoord;
7  in bool gl_FrontFacing;
8  in vec2 gl_PointCoord;
9  */
10 out vec4 color; // Output data
11
12 void main(){
13   color=fragmentColor;
14 }
```

Texture.VertexShader

```
1  #version 330 core
2
3  #define VERTEX_POSITION 0
4  #define TEXTURE_DATA 1
5
6  layout(location = VERTEX_POSITION)
7    in vec3 vertexPosition_modelspace;
8  layout(location = TEXTURE_DATA) in vec2 vertexUV;
9
10 out vec2 UV;
11
12 void main() {
13   gl_Position = vec4(vertexPosition_modelspace,1);
14   UV = vertexUV;
15 }
```

Texture.FragmentShader

```
1  #version 330 core
2
3  in vec2 UV;
4  uniform sampler2D myTextureSampler; // Texture data
5
6  out vec3 color; // Output data
7
8  void main(){
9   color = texture(myTextureSampler, UV).rgb;
10 }
```

void glBlendFunc(GLenum sfactor, GLenum dfactor)

$\cdot_s$ =shader output, $\cdot_d$ =framebuffer $R = R_s s_R + R_d d_R$
$G = G_s s_G + G_d d_G$
$B = B_s s_B + B_d d_B$
$A = A_s s_A + A_d d_A$

	(s_R, s_G, s_B, s_A) ja (d_R, d_G, d_B, d_A)
GL_ZERO	$(0, 0, 0, 0)$
GL_ONE	$(1, 1, 1, 1)$
GL_SRC_COLOR	(R_s, G_s, B_s, A_s)
GL_ONE_MINUS_SRC_COLOR	$(1 - R_s, 1 - G_s, 1 - B_s, 1 - A_s)$
GL_DST_COLOR	(R_d, G_d, B_d, A_d)
GL_ONE_MINUS_DST_COLOR	$(1 - R_d, 1 - G_d, 1 - B_d, 1 - A_d)$
GL_SRC_ALPHA	(A_s, A_s, A_s, A_s)
GL_ONE_MINUS_SRC_ALPHA	$(1 - A_s, 1 - A_s, 1 - A_s, 1 - A_s)$

void glBlendFunc(GLenum sfactor, GLenum dfactor)

	(s_R, s_G, s_B, s_A) ja (d_R, d_G, d_B, d_A)
GL_DST_ALPHA	(A_d, A_d, A_d, A_d)
GL_ONE_MINUS_DST_ALPHA	$(1 - A_d, 1 - A_d, 1 - A_d, 1 - A_d)$
GL_CONSTANT_COLOR	(R_c, G_c, B_c, A_c)
GL_ONE_MINUS_CONSTANT_COLOR	(R_c, G_c, B_c, A_c)
GL_CONSTANT_ALPHA	(A_c, A_c, A_c, A_c)
GL_ONE_MINUS_CONSTANT_ALPHA	$(1 - A_c, 1 - A_c, 1 - A_c, 1 - A_c)$
GL_SRC_ALPHA_SATURATE	$(\min(A_s, 1 - A_d), \min(A_s, 1 - A_d),$ $\min(A_s, 1 - A_d), 1)$

Muita läpikuultavuuteen liittyviä funktioita

- `glBlendColor(r, g, b, a)` – Asettaa arvot $\cdot_c$
- `glBlendFuncSeparate(...)` – Kuten `glBlendFunc`, mutta voidaan asettaa erilaiset kaavat väri- ja α-kanaville.
- `glBlendEquation(GLenum mode)` – "Vaihtaa $+$ -merkin muuksi operaatioksi" värien sekoituksessa.
- `glBlendEquation(GLenum mode)` – Kuten `glBlendEquation`, mutta väriarvoille ja α-kanavalle voidaan asetta oma operaattori.

Harjoituksia

(Ex 31) Laske edellisen esimerkin väriarvot pikseliin jossa $(R_s, G_s, B_s, A_s) = (1, 0.5, 0, 0.5)$ ja $(R_d, G_d, B_d, A_d) = (1, 1, 1, 0.75)$.

(Ex 32) Miten tulos muuttuu jos läpinäkyvyydelle käytetään asetusta `glBlendFunc(GL_SRC_ALPHA, GL_DST_ALPHA)`?

(Ex 33) Laske kahden edellisen tehtävän tulokset jos on annettu komento `glBlendEquation(GL_FUNC_SUBTRACT)`.

Harjoituksia

(Ex 34) Kirjoita ohjelma, jossa teksturoitu 8-kulmio sekä yksivärinen nelio liikkuvat ruudulla ennalta määrätyillä radoilla (esim ympyrää)

(Ex 35) Lisää neliöön läpinäkuultavuus (α-kanava).

(Ex 36) Muokkaa 8-kulmion piirtoa siten, että punaiset `rgb=vec3(1.0, 0.0, 0.0)` kohdat ovat läpinäkyviä. Vihje: Fragmentshader.

foo

- bar

foo

- bar