

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ SPOŁECZNY

„Image Processing and Computer Graphics”

Prezentacja multimedialna współfinansowana przez Unię Europejską w ramach Europejskiego Funduszu Społecznego w projekcie pt. „Innowacyjna dydaktyka bez ograniczeń - zintegrowany rozwój Politechniki Łódzkiej - zarządzanie Uczelnią, nowoczesna oferta edukacyjna i wzmacniania zdolności do zatrudniania osób niepełnosprawnych”

Politechnika Łódzka, ul. Żeromskiego 116, 90-924 Łódź, tel. (042) 631 28 83
www.kapitalludzki.p.lodz.pl

Contribution Image Processing & Computer Graphics 2

Lecturer:
Piotr M. Szczypiński, Dr inż.
Institute of Electronics, Technical University of Lodz
Wolczanska 211/215, 90-924 Lodz, Poland
tel. +48426312638
<http://www.elel.p.lodz.pl/pms>
pms@p.lodz.pl

Fragments of the lecture were also provided by
Andrzej Materka, Michał Strzelecki, Paweł Strumiłło,
Piotr Romaniuk, Piotr Makowski
and were used with permission.

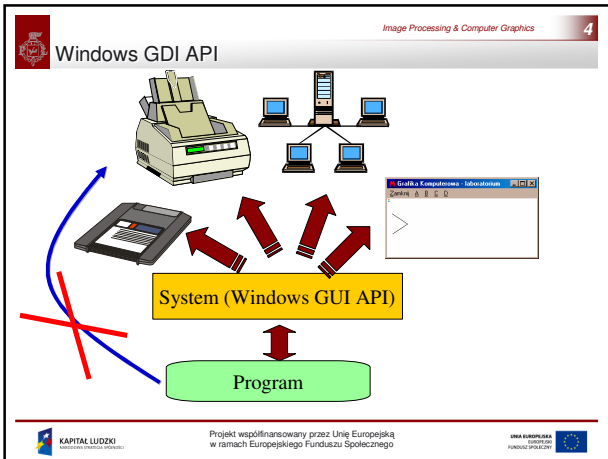
Portions of this lecture are illustrated with articles from the wikipedia
(GNU Free Documentation License
http://en.wikipedia.org/wiki/Wikipedia:Text_of_the_GNU_Free_Documentation_License)

Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego

Programming Image Processing & Computer Graphics 3

- Windows 2D graphics API
- Concept of Context
- 3D programming technologies (Direct3D vs. OpenGL)
- OpenGL functions
- OpenGL context
- Scene, lights and shading
- Transformations and matrix stack
- Rendering objects
- Tutorials

Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego



Windows and pixmaps are collectively named drawables, and their content data resides on the server. A client can however request the content of a drawable to be transferred from the server to the client or vice versa.

Graphic contexts and fonts

The client can request a number of graphic operations, such as clearing an area, copying an area into another, drawing points, lines, rectangles, and text. Beside clearing, all operations are possible on all drawables, both windows and pixmaps.

Most requests for graphic operations include a graphic context, which is a structure that contains the parameters of the graphic operations. A graphic context includes the foreground color, the background color, the font of text, and other graphic parameters. When requesting a graphic operation, the client includes a graphic context. Not all parameters of the graphic context affect the operation; for example, the font does not affect drawing a line.

The core protocol specifies the use of server-side fonts.^[1] Such fonts are stored as files, and the server accesses them either directly via the local filesystem or via the network from another program called font server. Clients can request the list of fonts available to the server and can request a font to be loaded (if not already) or unloaded (if not used by other clients) by the server. A client can request general information about a font (for example, the font ascent) and the space a specific string takes when drawn with a specific font.

The names of the fonts are arbitrary strings at the level of the X Window core protocol. The X logical font description conventions^[2] specify how fonts should be named according to their attributes. These conventions also specify the values of optional properties that can be attached to fonts.

The `x11rcore1` program prints the list of fonts stored in the server. The `x11rcore1` program shows the glyphs of fonts, and allows the user to select the name of a font for pasting it in another window.

The use of server-side fonts is currently considered deprecated in favour of client-side fonts.^[3] Such fonts are rendered by the client, not by the server, with the support of the `XR` or `cairo` libraries and the `XRrender` extension. No specification on client-side fonts is given in the core protocol.

Technical details

A Device Context (DC) is used to define the attributes of text and images that are output to the screen or printer. The actual context is maintained by GDI. An HDC, which is a handle to the structure, is obtained before output is written and released after the elements have been written.

A DC, like most GDI objects, is opaque - its data cannot be accessed directly, but its handle can be passed to various GDI functions that will operate on it, either to draw an object, to retrieve information about it, or to change the object in some way.

GDI+

With the introduction of Windows XP, GDI was deprecated in favor of its successor, the C++ based GDI+ subsystem. Unlike its predecessor GDI which did not access the graphics hardware directly, GDI+ provided hardware acceleration by interacting directly with the graphics device on behalf of the application.^[1] GDI+ adds anti-aliased 2D graphics, floating point coordinates, gradient shading, more complex path management, intrinsic support for modern graphics file formats like JPEG and PNG, and support for composition of affine transformations in the 2D view pipeline. GDI+ uses ARGB values to represent color. Use of these features is apparent in Windows XP's user interface and several of its applications such as Microsoft Paint, Windows Picture and Fax Viewer, Photo Printing Wizard, My Pictures Slideshow screensaver, and their presence in the basic graphics layer greatly simplifies implementations of vector-graphics systems such as Flash or SVG. The GDI+ dynamic library can be shipped with an application and used under older versions of Windows.

Because of the additional text processing and resolution independence capabilities in GDI+, text rendering is nearly an order of magnitude slower than in GDI.^[2] Chris Jackson, an application compatibility expert working for Microsoft published some tests indicating that a piece of text rendering code he had written could render 99 000 glyphs per second in GDI, but the same code using GDI+ rendered 16 600 glyphs per second.

The Microsoft .NET class library provides a managed interface for GDI+ via the `System.Drawing` namespace.

GDI+ is similar (in purpose and structure) to Apple's Quartz 2D subsystem, and the open-source `libart` and `Cairo` libraries.

GDI and GDI+ applications in Windows Vista

Starting with Windows Vista, all Windows applications including GDI and GDI+ applications run in the new compositing engine, `Direct3D Windows Presentation Subsystem` which is built atop the `Microsoft Windows Presentation Model`. The GDI render path is redirected through `WMR` and

The screenshot shows the MSN Accessibility Developer Center website. The main content area is titled "Windows GDI Start Page". It includes a "Purpose" section explaining that GDI enables applications to use graphics and formatted text on video displays and printers. A "Where Applicable" section states that GDI is used in all Windows-based applications. A "Developer Audience" section notes that the API is designed for C/C++ programmers familiar with the Windows graphical user interface and message-driven architecture. The left sidebar contains a navigation menu with categories like "Azure Services Platform", "Design Tools", and "Graphics and Multimedia".

The screenshot shows the "About Device Contexts" page on the MSN Accessibility Developer Center. It explains that device independence is a key feature of Microsoft Windows, with applications drawing and printing on various devices. It details how device contexts (DCs) are defined in dynamic-link libraries (DLLs) and how they interact with device drivers. The page lists topics covered in the overview, such as Graphic Objects, Graphic Modes, Device Context Types, Device Context Operations, and ICH-Enabled Device Context Functions. It also mentions the layout of a DC or a window and refers to "Window Layout and Mirroring" for more information.

Image Processing & Computer Graphics 9

How to create and delete context

```
// Window redrawing context
HDC context = BeginPaint(window, &rect);
// Drawing functions come here
EndPaint(window, &rect);

// Printer context
PRINTDLG pd;
PrintDlg(&pd);
HDC context = pd.hDC;
StartPage(context);
// Drawing functions come here
EndPage(context);
DeleteDC(context);

// Windows metafile context
HDC context = CreateEnhMetaFile(NULL, "file.emf", NULL, NULL);
// Drawing functions come here
CloseEnhMetaFile(context);
```

KAPITAŁ LUDZKI
INICJATYWA WZROSTAJĄCYCH REGIONÓW

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
ROZWOJ
FINANSOWANIE SPÓŁNOCNE

Image Processing & Computer Graphics 10

Modifying context

```

HPEN pen;
HBRUSH brush;
HGDIOBJ oldpen, oldbrush;
pen = CreatePen(PS_SOLID, 1, RGB(rp, gp, bp)); // rp, gp, bp są skład. kolorów
brush = CreateSolidBrush(RGB(rb, gb, bb)); // rb, gb, bb są skład. kolorów
oldpen = SelectObject(context, pen);
oldbrush = SelectObject(context, brush);
...
// Drawing functions come here
...
SelectObject(context, oldbrush);
SelectObject(context, oldpen);
DeleteObject(brush);
DeleteObject(pen);

```

KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA
FUNDUSZ SPOŁECZNY

Image Processing & Computer Graphics 11

GDI drawing functions

```

SetPixelV(context, x, y, RGB(r, g, b));
MoveToEx(context, x, y, NULL);
LineTo(context, x, y);
Rectangle(context, x1, y1, x2, y2);
TextOut(context, x, y, "Napis", z);
StretchDIBits(context,
    win_left, win_left, win_right, win_bottom,
    bmp_left, bmp_left, bmp_right, bmp_bottom,
    bmp_pointer, header_pointer,
    DIB_RGB_COLORS, SRCCOPY);

HENHMETAFILE hemf = GetEnhMetaFile("file.emf");
PlayEnhMetaFile(context, hemf, &rect);
DeleteEnhMetaFile(hemf);

```

KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA
FUNDUSZ SPOŁECZNY

The screenshot shows the MSDN website's 'Windows GDI Start Page'. The browser's address bar shows the URL: http://msdn.microsoft.com/en-us/library/6d14c202%28VS.85%29.aspx. The page has a red header with the 'msdn' logo and a search bar. Below the header is a navigation bar with 'Home', 'Library', 'Learn', 'Downloads', 'Support', and 'Community'. A left sidebar contains a tree view of GDI-related topics. The main content area has a breadcrumb trail: 'MSDN > MSDN Library > Win32 and COM Development > Graphics and Multimedia > Windows GDI'. The 'Windows GDI Start Page' title is followed by a 'Purpose' section, which states: 'The Microsoft® Windows® graphics device interface (GDI) enables applications to use graphics and formatted text on both the video display and the printer. Windows-based applications do not access the graphics hardware directly. Instead, GDI interacts with device drivers on behalf of applications.' Below this are sections for 'Where Applicable' and 'Developer Audience'.

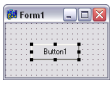
Image Processing & Computer Graphics 16

Libraries which takes over (C++ Builder)

```

#include <vccl.h>
#pragma hdrstop
#include "Unit1.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
}
//-----
void __fastcall TForm1::ButtonClick(TObject* Sender)
{
}
MessageBeep( 1);
}

```

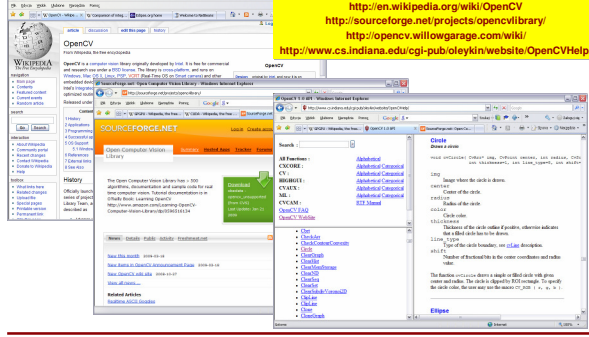


Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

Image Processing & Computer Graphics 17

OpenCV for image processing

<http://en.wikipedia.org/wiki/OpenCV>
<http://sourceforge.net/projects/opencvlibrary/>
<http://opencv.willowgarage.com/wiki/>
<http://www.cs.indiana.edu/cgi-pub/oleykin/website/OpenCVHelp/>



Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

Image Processing & Computer Graphics 18

OpenCV functions

- Naming convention:

<div style="display: flex; flex-direction: column; align-items: center;"> <div style="margin-bottom: 5px;">cvCircle(...)</div> <div style="display: flex; justify-content: center; gap: 10px;"> <div style="border-top: 1px solid black; width: 20px;"></div> <div style="border-top: 1px solid black; width: 20px;"></div> <div style="border-top: 1px solid black; width: 20px;"></div> </div> <div style="display: flex; justify-content: center; gap: 10px; margin-top: 5px;"> <div style="writing-mode: vertical-rl; transform: rotate(180deg); font-size: small;">library prefix</div> <div style="writing-mode: vertical-rl; transform: rotate(180deg); font-size: small;">name core</div> </div> </div>	<p>OpenCV libraries:</p> <ul style="list-style-type: none"> cxcore – basic structures and functions cv – image processing functions cvaux – auxiliary library highgui – user interface functions ml – machine learning <p>C/C++ header files: cv.h, cvaux.h, cxcore.h, ml.h, cvcam.h, highgui.h</p> <p>Binary libraries (Windows): cv.lib, cvaux.lib, cvcam.lib, cvhaartraining.lib, cxcore.lib, cxts.lib, highgui.lib, ml.lib</p>
---	---

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

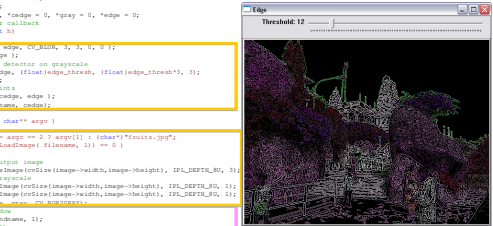
Image Processing & Computer Graphics 19

OpenCV programming example

```
#include "cv.h"
#include "highgui.h"
char window[] = "Edge";
char trackbar[] = "Threshold";
int edge_thresh = 1;
IplImage *image = 0, *edge = 0, *gray = 0, *edge = 0;
// define a trackbar callback
void on_trackbar(int) {
}

// Sobel's gray edge, Cx3Sb(1, 3, 3, 1, 0)
cvSobel(gray, edge, CV_32S, 1, 3, 3, 1, 0);
// Run the edge detection on grayscale
cvCanny(gray, edge, (float)edge_thresh, (float)edge_thresh, 3);
cvCvtColor(image, gray, CV_BGR2GRAY);
// copy edge points
cvCopy(edge, edge, edge);
cvShowImage(window, edge);

int main(int argc, char** argv) {
char* filename = argv == 2 ? argv[1] : (char*)"results.jpg";
if (!image = cvLoadImage(filename, 1)) return -1;
// Create an output image
edge = cvCreateImage(cvSize(image->width, image->height), IPL_DEPTH_8U, 3);
// Convert to grayscale
gray = cvCreateImage(cvSize(image->width, image->height), IPL_DEPTH_8U, 1);
edge = cvCreateImage(cvSize(image->width, image->height), IPL_DEPTH_8U, 3);
// Show the image
cvNamedWindow(window, 1);
cvSetTrackbarPos(trackbar, window, edge_thresh, 100, on_trackbar);
// Show the image
cvShowImage(window);
// Wait for a key stroke; the same function arranges events processing
cvWaitKey(0);
cvReleaseImage(&image);
cvReleaseImage(&gray);
cvReleaseImage(&edge);
return 0;
}
```

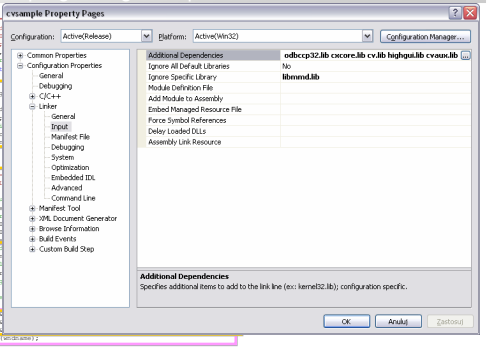


OpenCV sample code edge.c

KAPITAL LUDZKI
Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA
FUNDUSZ SPÓŁECZNY

Image Processing & Computer Graphics 20

OpenCV programming example



KAPITAL LUDZKI
Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA
FUNDUSZ SPÓŁECZNY

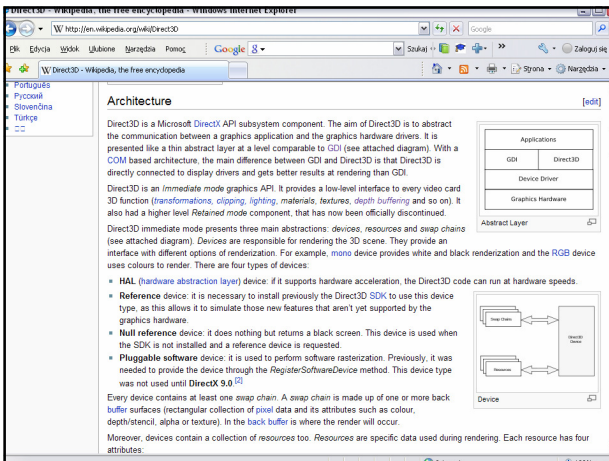
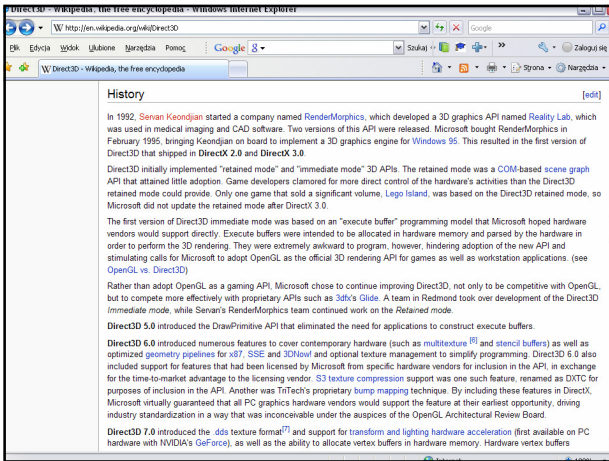
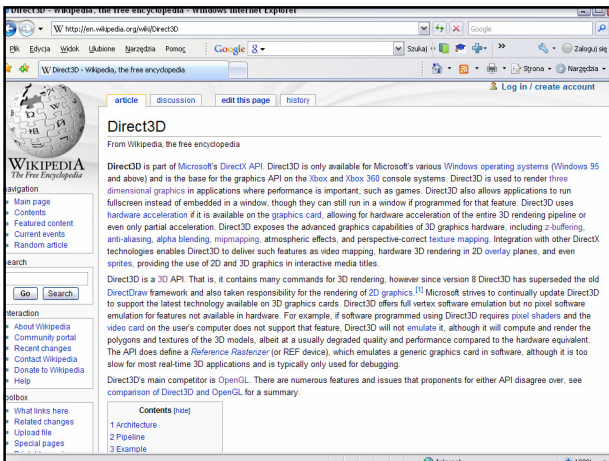
Image Processing & Computer Graphics 21

Three dimensional graphics API

Direct3D

OpenGL

KAPITAL LUDZKI
Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA
FUNDUSZ SPÓŁECZNY



Pipeline

The Microsoft Direct3D 10 API defines a process to convert a group of vertices, textures, buffers, and state into an image on the screen. This process is described as a rendering pipeline with several distinct stages. The different stages of the Direct3D 10 pipeline^[4] are:^[5]

- Input Assembler:** Reads in vertex data from an application supplied vertex buffer and feeds them down the pipeline.
- Vertex Shader:** Performs operations on a single vertex at a time, such as transformations, skinning, or lighting.
- Geometry Shader:** Processes entire primitives such as triangles, points, or lines. Given a primitive, this stage discards it, or generates one or more new primitives.
- Stream Output:** Can write out the previous stage's results to memory. This is useful to recirculate data back into the pipeline.
- Rasterizer:** Converts primitives into pixels, feeding these pixels into the pixel shader. The Rasterizer may also perform other tasks such as clipping what is not visible, or interpolating vertex data into per-pixel data.
- Pixel Shader:** Determines the final pixel colour to be written to the render target and can also calculate a depth value to be written to the depth buffer.
- Output Merger:** Merges various types of output data (pixel shader values, alpha blending, depth/stencil, ...) to build the final result.

The pipeline stages illustrated with a round box are fully programmable. The application provides a shader program that describes the exact operations to be completed for that stage. Many stages are optional and can be disabled altogether.

Example

Drawing a triangle in Direct3D:

```

// A 3-vertex polygon definition

```

OpenGL

From Wikipedia, the free encyclopedia

It has been suggested that *IRIS GL* be merged into this article or section. (Discuss)

OpenGL (**Open Graphics Library**) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992^[4] and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation. It is also used in video games, where it competes with Direct3D on Microsoft Windows platforms (see Direct3D vs. OpenGL). OpenGL is managed by the non-profit technology consortium, the Khronos Group.

Contents [hide]

- 1 Specification
- 2 Design
- 3 Example
- 4 Documentation
- 5 Extensions
- 6 Associated utility libraries
- 7 Bindings
- 8 Higher level functionality
- 9 History
- 10 OpenGL 2.0
- 11 OpenGL 2.1

History

In the 1980s, developing software that could function with a wide range of graphics hardware was a real challenge. Software developers wrote custom interfaces and drivers for each piece of hardware. This was expensive and resulted in much duplication of effort. By the early 1990s, Silicon Graphics (SGI) was a leader in 3D graphics for workstations. Their *IRIS GL* API^[4] was considered the state of the art and became the de facto industry standard, overshadowing the open standards-based PHIGS. This was because IRIS GL was considered easier to use, and because it supported *immediate mode* rendering. By contrast, PHIGS was considered difficult to use and outdated in terms of functionality.

SGI's competitors (including Sun Microsystems, Hewlett-Packard and IBM) were also able to bring to market 3D hardware, supported by extensions made to the PHIGS standard. This in turn caused SGI market share to weaken as more 3D graphics hardware suppliers entered the market. In an effort to influence the market, SGI decided to turn the IrisGL API into an open standard. SGI considered that the IrisGL API itself wasn't suitable for opening due to licensing and patent issues. Also, the IrisGL had API functions that were not relevant to 3D graphics. For example, it included a windowing, keyboard and mouse API, in part because it was developed before the X Window System and Sun's NeWS systems were developed.

In addition, SGI had a large number of software customers; by changing to the OpenGL API they planned to keep their customers locked onto SGI (and IBM) hardware for a few years while market support for OpenGL matured. Meanwhile, SGI would continue to try to maintain their customers tied to SGI hardware by developing the advanced and proprietary *Iris Inventor* and *Iris Performer* programming APIs.

As a result, SGI released the **OpenGL** standard.

The OpenGL standardised access to hardware, and pushed the development responsibility of hardware interface programs, sometimes called *device drivers*, to hardware manufacturers and delegated windowing functions to the underlying operating system. With so many different kinds of graphic hardware, getting them all to speak the same language in this way had a remarkable impact by giving software developers a higher level platform for 3D software development.

In 1992,^[4] SGI led the creation of the OpenGL architectural review board (OpenGL ARB), the group of companies that would maintain and expand the OpenGL specification for years to come. OpenGL evolved from (and is very similar in style to) SGI's earlier 3D interface, *IrisGL*. One of the restrictions of IrisGL was that it only provided access to features supported by the underlying hardware. If the graphics hardware did not support a feature, then the application could not use it. OpenGL overcame this problem by providing

History

In the 1980s, developing software that could function with a wide range of graphics hardware was a real challenge. Software developers wrote custom interfaces and drivers for each piece of hardware. This was expensive and resulted in much duplication of effort. By the early 1990s, Silicon Graphics (SGI) was a leader in 3D graphics for workstations. Their *IRIS GL* API^[4] was considered the state of the art and became the de facto industry standard, overshadowing the open standards-based PHIGS. This was because IRIS GL was considered easier to use, and because it supported *immediate mode* rendering. By contrast, PHIGS was considered difficult to use and outdated in terms of functionality.

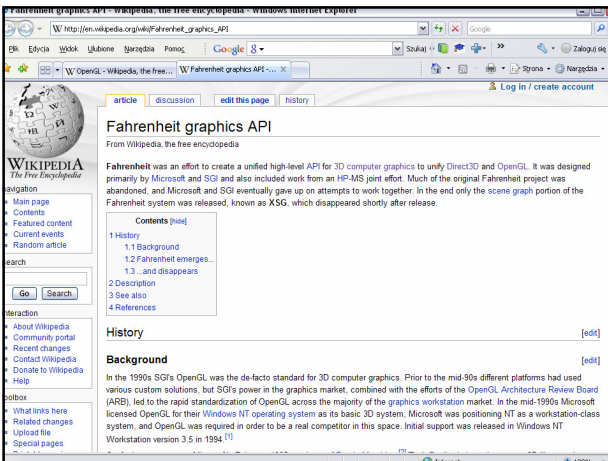
SGI's competitors (including Sun Microsystems, Hewlett-Packard and IBM) were also able to bring to market 3D hardware, supported by extensions made to the PHIGS standard. This in turn caused SGI market share to weaken as more 3D graphics hardware suppliers entered the market. In an effort to influence the market, SGI decided to turn the IrisGL API into an open standard. SGI considered that the IrisGL API itself wasn't suitable for opening due to licensing and patent issues. Also, the IrisGL had API functions that were not relevant to 3D graphics. For example, it included a windowing, keyboard and mouse API, in part because it was developed before the X Window System and Sun's NeWS systems were developed.

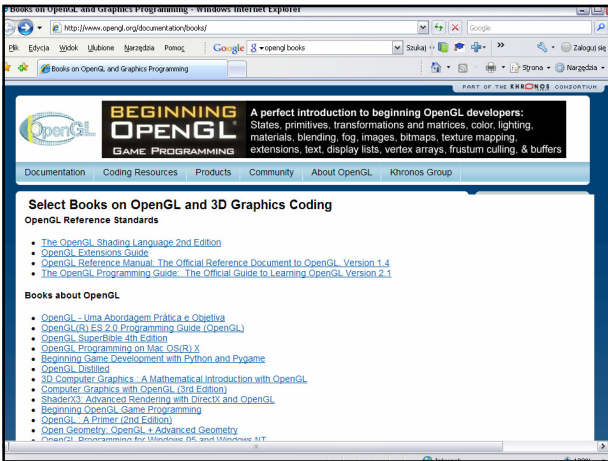
In addition, SGI had a large number of software customers; by changing to the OpenGL API they planned to keep their customers locked onto SGI (and IBM) hardware for a few years while market support for OpenGL matured. Meanwhile, SGI would continue to try to maintain their customers tied to SGI hardware by developing the advanced and proprietary *Iris Inventor* and *Iris Performer* programming APIs.

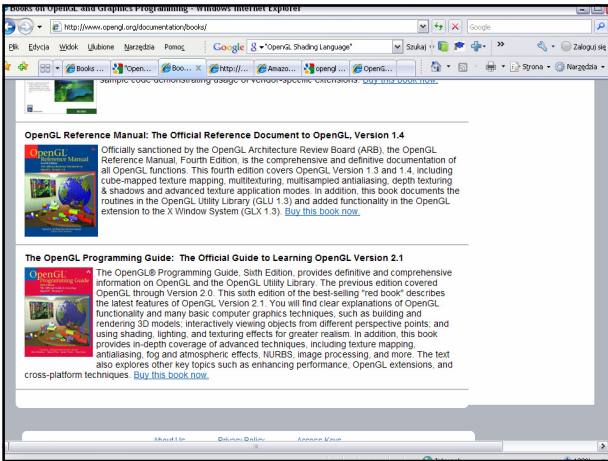
As a result, SGI released the **OpenGL** standard.

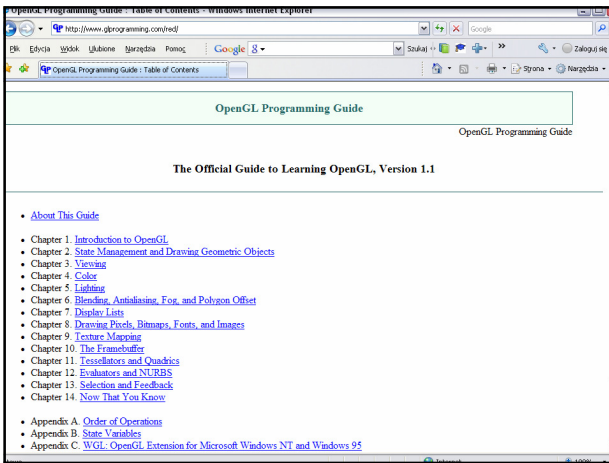
The OpenGL standardised access to hardware, and pushed the development responsibility of hardware interface programs, sometimes called *device drivers*, to hardware manufacturers and delegated windowing functions to the underlying operating system. With so many different kinds of graphic hardware, getting them all to speak the same language in this way had a remarkable impact by giving software developers a higher level platform for 3D software development.

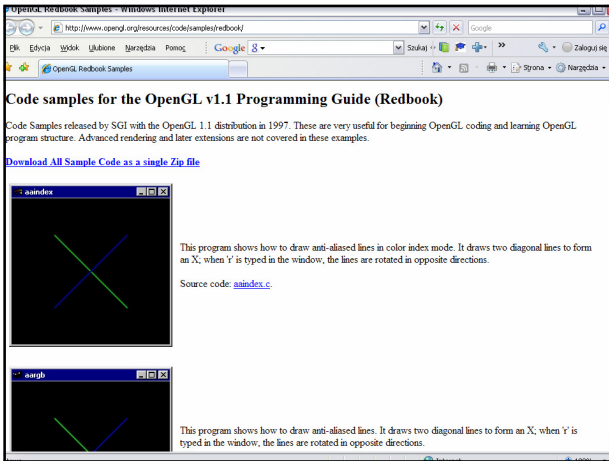
In 1992,^[4] SGI led the creation of the OpenGL architectural review board (OpenGL ARB), the group of companies that would maintain and expand the OpenGL specification for years to come. OpenGL evolved from (and is very similar in style to) SGI's earlier 3D interface, *IrisGL*. One of the restrictions of IrisGL was that it only provided access to features supported by the underlying hardware. If the graphics hardware did not support a feature, then the application could not use it. OpenGL overcame this problem by providing

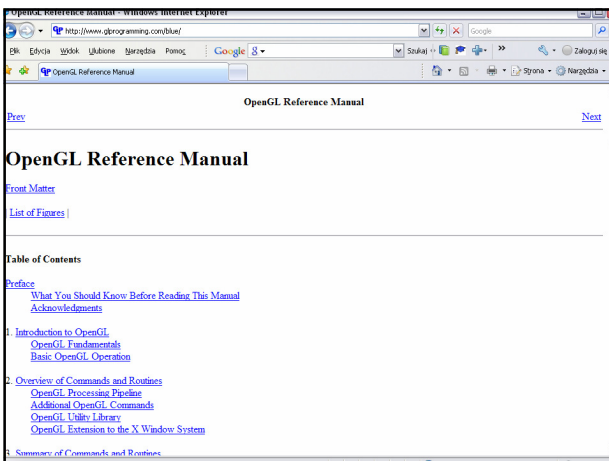












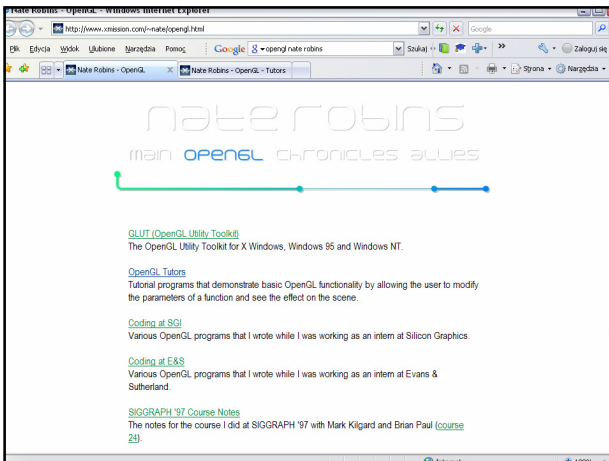




Image Processing & Computer Graphics 36

OpenGL functions

- Naming convention:

library prefix

name core

number of arguments

argument type suffix

glColor3fv(...)

Prefix:

- xgl, wgl, cgl* – platform-specific frameworks
- gl* – general OpenGL functions
- glu* – utility functions, useful routines, a layer above gl
- glut* – utility toolkit functions
- aux* – auxiliary library (deprecated)
- glui* – user interface functions

Other libraries also exist.

C/C++ header files: *gl.h, glu.h, glut.h, aux.h*
 Binary libraries (Windows): *opengl32.dll, glu32.dll, glut32, glaux.lib*

Suffix: *d* – double, *f* – float, *i* – integer, *s* – short, *v* – vector

KAPITAŁ LUDZKI
WARSZAWA UNIWERSYTET WYDZIAŁ

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
ROZWÓJ I WYKONANIE
FINANSOWANIE

Image Processing & Computer Graphics 37

Example (creating a window)

Windows specific

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpCmdLine, int nCmdShow)
{
MSG komunikat;
WNDCLASS klasaOkna;
HMENU okno;
klasaOkna.style = CS_HREDRAW | CS_VREDRAW;
klasaOkna.lpfnWndProc = (WNDPROC) WinProc;
klasaOkna.cbClsExtra = 0;
klasaOkna.cbWndExtra = 0;
klasaOkna.hInstance = hInstance;
klasaOkna.hIcon = NULL;
klasaOkna.hCursor = LoadCursor(NULL, IDC_ARROW);
klasaOkna.hbrBackground = NULL;
klasaOkna.lpszMenuName = NULL;
klasaOkna.lpszClassName = "Okno_OpenGL";
if (RegisterClass(&klasaOkna) == 0) return 0;
okno = CreateWindow("Okno_OpenGL", "Okno OpenGL",
WS_OVERLAPPEDWINDOW|WS_CLIPCHILDREN|WS_CLIPSIBLINGS,
100, 100, 250, 250,
NULL, NULL, hInstance, NULL);
if (!okno) return 1;
ShowWindow(okno, nCmdShow);
UpdateWindow(okno);
while ( GetMessage(&komunikat, NULL, 0, 0) )
{
TranslateMessage(&komunikat);
DispatchMessage(&komunikat);
}
return 0;
}
```

Portable
preferable is glui
aux is deprecated

```
int WINAPI WinMain(
HINSTANCE hInstance,
HINSTANCE hPrevInstance,
LPSTR lpCmdLine,
int nCmdShow)
{
// Create a window
auxInitDisplayMode(AUX_DOUBLE | AUX_RGBA);
auxInitPosition(100, 100, 250, 250);
auxInitWindow("Okno OpenGL");
// Window resize handler
auxReshapeFunc(ZmianaRozmiarOkna);
// Keyboard event handlers
auxKeyFunc(AUX_LEFT, StrzalkaLewo);
auxKeyFunc(AUX_RIGHT, StrzalkaPrawo);
auxKeyFunc(AUX_DOWN, StrzalkaDol);
auxKeyFunc(AUX_UP, StrzalkaGora);
// Calling function for setting
// a background, lights, etc.
UstawScene();
// Loop for event handling
auxMainLoop(RenderujScene);
}
```

KAPITAL LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego



Image Processing & Computer Graphics 38

Example (event handling)

Windows specific

```
long WINAPI WinProc(HWND okno, UINT kmdk, WPARAM wparam, LPARAM lparam)
{
switch ( kmdk )
{
case WM_CREATE:
{
kontekst_okna = GetDC(okno);
UstawFormatGrafiki();
kontekst_opengl = wglCreateContext(kontekst_okna);
wglMakeCurrent(kontekst_okna, kontekst_opengl);
UstawScene();
}
return 0;
case WM_DESTROY:
{
wglMakeCurrent(kontekst_okna, NULL);
wglDeleteContext(kontekst_opengl);
PostQuitMessage(0);
return 0;
case WM_SIZE:
{
ZmianaRozmiarOkna(LWORD(lparam), HWORD(lparam));
return 0;
case WM_KEYDOWN:
{
switch (lparam)
{
case VK_F5: StrzalkaDol(); break;
case VK_LEFT: StrzalkaLewo(); break;
case VK_RIGHT: StrzalkaPrawo(); break;
case VK_DOWN: StrzalkaDol(); break;
}
SwapBuffers(kontekst_okna);
return 0;
case WM_PAINT:
{
RenderujScene();
SwapBuffers(kontekst_okna);
ValidateRect(okno, NULL);
break;
}
}
return DefWindowProc(okno, kmdk, wparam, lparam);
}
```

```
void CALLBACK StrzalkaLewo(void)
{
yRot -= 5.0f;
RenderujScene();
}
void CALLBACK StrzalkaPrawo(void)
{
yRot += 5.0f;
RenderujScene();
}
void CALLBACK StrzalkaDol(void)
{
xRot -= 5.0f;
RenderujScene();
}
void CALLBACK StrzalkaGora(void)
{
xRot += 5.0f;
RenderujScene();
}
```

KAPITAL LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego



Image Processing & Computer Graphics 39

Example (setting up the context)

Windows specific

```
HGLRC kontekst_opengl;
(...)
case WM_CREATE:
{
kontekst_okna = GetDC(okno);
UstawFormatGrafiki();
kontekst_opengl = wglCreateContext(kontekst_okna);
wglMakeCurrent(kontekst_okna, kontekst_opengl);
UstawScene();
return 0;
}
void UstawFormatGrafiki(void)
{
static PIXELFORMATDESCRIPTOR pfd =
{
sizeof(PIXELFORMATDESCRIPTOR),
1,
PFD_DRAW_TO_WINDOW | PFD_DOUBLEBUFFER | PFD_SUPPORT_OPENGL,
PFD_TF_RGB, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 16, 0, 0,
PFD_MAIN_PLANE,
0, 0, 0
};
SetPixelFormat(kontekst_okna, ChoosePixelFormat(kontekst_okna, &pfd), &pfd);
}
```

Windows specific

```
static PIXELFORMATDESCRIPTOR pfd =
{
sizeof(PIXELFORMATDESCRIPTOR),
1,
PFD_DRAW_TO_WINDOW | PFD_DOUBLEBUFFER | PFD_SUPPORT_OPENGL,
PFD_TF_RGB, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 16, 0, 0,
PFD_MAIN_PLANE,
0, 0, 0
};
SetPixelFormat(kontekst_okna, ChoosePixelFormat(kontekst_okna, &pfd), &pfd);
}
```

KAPITAL LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego





Example (setting up the world)

```
void UstawScene()
{
    // Coordinates and colors of lights
    GLfloat ambientLight[] = {0.4f, 0.4f, 0.4f, 1.0f };
    GLfloat diffuseLight[] = {0.7f, 0.7f, 0.7f, 1.0f };
    GLfloat specular[] = { 0.9f, 0.9f, 0.9f, 1.0f};
    GLfloat lightPos[] = { -500.0f, 2000.0f, 2000.0f, 1.0f };
    GLfloat specref[] = { 0.6f, 0.6f, 0.6f, 1.0f };
    glEnable(GL_DEPTH_TEST); // Z-buffer check on
    glEnable(GL_CULL_FACE); // Face culling on
    glEnable(GL_LIGHTING); // Shading models on
    glEnable(GL_NORMALIZE); // Computation of normal vectors on
    glEnable(GL_AUTO_NORMAL);
    // Lights setup
    glEnable(GL_LIGHT_MODEL_AMBIENT,ambientLight);
    glLightfv(GL_LIGHT0,GL_AMBIENT,ambientLight);
    glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuseLight);
    glLightfv(GL_LIGHT0,GL_SPECULAR,specular);
    glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
    glEnable(GL_LIGHT0);
    // Shading models setup
    glEnable(GL_COLOR_MATERIAL);
    glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
    glMaterialfv(GL_FRONT, GL_SPECULAR,specref);
    glMateriali(GL_FRONT, GL_SHININESS,64);
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Background color
}
```



Nate Robins Tutors (Light & Material, Light Position)

The screenshot shows a tutorial window with several panels. On the left, there's a 3D view of a soccer ball. In the center, there are code snippets for setting up lights and materials. On the right, there's another 3D view showing the light position relative to the ball. Below the code, there are sliders for adjusting light parameters.

```
GLfloat light_pos[] = { -2.00, 2.00, -2.00, 1.00 };
GLfloat light_Kd[] = { 0.00, 0.00, 0.00, 1.00 };
GLfloat light_Ks[] = { 1.00, 1.00, 1.00, 1.00 };
GLfloat light_Kd[] = { 1.00, 1.00, 1.00, 1.00 };
GLfloat material_Kd[] = { 0.11, 0.00, 0.11, 1.00 };
GLfloat material_Ks[] = { 0.43, 0.47, 0.54, 1.00 };
GLfloat material_Kd[] = { 0.33, 0.33, 0.32, 1.00 };
GLfloat material_Ks[] = { 0.00, 0.00, 0.00, 0.00 };
GLfloat material_Spec = 0;
```

```
GLfloat pos[] = { 1.50, 1.00, 1.00, 0.00 };
GLfloat look[] = { 0.00, 0.00, 2.00, -0.00 };
GLfloat eye[] = { 0.00, 0.00, 0.00, 0.00 };
GLfloat center[] = { 0.00, 0.00, 0.00, 0.00 };
GLfloat up[] = { 0.00, 1.00, 0.00, 0.00 };
glLightfv(GL_LIGHT0, GL_POSITION, pos);
```



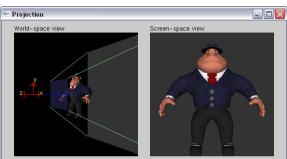
Example (setting up frustum)

```
void CALLBACK ZmianaRozmiaruOkna(GLsizei w, GLsizei h)
{
    glViewport(0, 0, w, h);
    // Projection matrix stack initialization
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    // Setting up frustum
    glOrtho (-nRange*w/h, nRange*w/h, -nRange, nRange, -nRange*2.0f, nRange*2.0f);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
// Setting up frustum
glOrtho(left, right, bottom, top, near, far)
glFrustum(left, right, bottom, top, near, far)
// Don't like glOrtho and glFrustum?
gluPerspective(90, width / height, 1, 100);
gluLookAt(10.0, 10.0, 0.0, // camera location
0.0, 0.0, -30.0, // looking at
0.0, 1.0, 0.0); // up vector
```

See also: <http://www.giprogramming.com/blue/>

Image Processing & Computer Graphics **43**

Nate Robins Tutors (Projection)




Control manipulation window

```

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

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



Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego


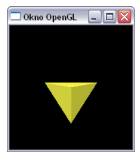



Image Processing & Computer Graphics **44**

Example (Rendering)



```

void CALLBACK RenderJScene(void)
{
    // Clear window
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // Save transformation matrix
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glRotatef(xRot, 1.0f, 0.0f, 0.0f);
    glRotatef(yRot, 0.0f, 1.0f, 0.0f);
    // Define object (here triangle faces)
    glBegin(GL_TRIANGLES);
    glFrontFace(GL_CW);
    glColor3f(0.2f, 0.2f, 0.8f);
    glNormal3f(0.58f, 0.58f, 0.58f);
    glVertex3f(0.0f, 60.0f, 0.0f);
    glVertex3f(60.0f, 0.0f, 0.0f);
    glVertex3f(0.0f, 0.0f, 60.0f);
    (...)
    glNormal3f(0.0f, -1.0f, 0.0f);
    glVertex3f(0.0f, 0.0f, 60.0f);
    glVertex3f(60.0f, 0.0f, 0.0f);
    glVertex3f(0.0f, 0.0f, 0.0f);
    glEnd();
    glPopMatrix();
    glFlush();
}
    
```



Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego


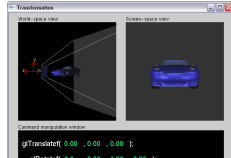


Image Processing & Computer Graphics **45**

Nate Robins Tutors (Transformation, Shapes)

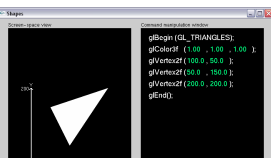


Control manipulation window

```

glTranslatef( 0.00 , 0.00 , 0.00 );
glRotatef( 0.0 , 0.00 , 1.00 , 0.00 );
glScalef( 1.00 , 1.00 , 1.00 );
glBegin( ... );
    
```

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




Control manipulation window

```

glBegin( GL_TRIANGLES );
glColorf( 1.00 , 1.00 , 1.00 );
glVertexf( 100.0 , 50.0 );
glVertexf( 50.0 , 150.0 );
glVertexf( 200.0 , 200.0 );
glEnd();
    
```

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



Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego




Image Processing & Computer Graphics 46

Transformation and matrix stack

```

// Projection CCS to FCS
glMatrixMode(GL_PROJECTION);
glPushMatrix();
glRotatef(kat_x, 1.0f, 0.0f, 0.0f);
glRotatef(kat_y, 0.0f, 1.0f, 0.0f);
glRotatef(kat_z, 0.0f, 0.0f, 1.0f);
glTranslatef(x, y, z);

(...)

glMatrixMode(GL_PROJECTION);
glPopMatrix();

glmMatrixMode(GL_MODELVIEW);
// Transformation SCS to CCS
glPushMatrix();
glLoadIdentity();
glRotatef(kat_x, 1.0f, 0.0f, 0.0f);
glRotatef(kat_y, 0.0f, 1.0f, 0.0f);
glRotatef(kat_z, 0.0f, 0.0f, 1.0f);
glTranslatef(x, y, z);

// Model 1 transformation
glPushMatrix();
glRotatef(...);
glTranslatef(...);
//Rendering of model 1 here
glPopMatrix();

// Model 2 transformation
glPushMatrix();
glRotatef(...);
glTranslatef(...);
//Rendering of model 2 here
glPopMatrix();

```

(...) ←

KAPITAL LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA FUNDUSZ SPÓŁECZNY

Image Processing & Computer Graphics 47

Transformation and matrix stack

glPushMatrix, **glPopMatrix** - push and pop the current matrix stack

There is a stack of matrices for each of the matrix modes. In **GL_MODELVIEW** mode, the stack depth is at least 32. In the other two modes, **GL_PROJECTION** and **GL_TEXTURE**, the depth is at least 2. The current matrix in any mode is the matrix on the top of the stack for that mode. **glPushMatrix** pushes the current matrix stack down by one, duplicating the current matrix. That is, after a **glPushMatrix** call, the matrix on the top of the stack is identical to the one below it. **glPopMatrix** pops the current matrix stack, replacing the current matrix with the one below it on the stack. Initially, each of the stacks contains one matrix, an identity matrix.

<http://www.glprogramming.com/blue/>

KAPITAL LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA FUNDUSZ SPÓŁECZNY

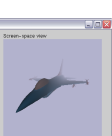
Image Processing & Computer Graphics 48

Effects (Textures, Fog)

GLfloat border_color[] = { 1.00, 0.00, 0.00, 1.00 };
GLfloat env_color[] = { 0.00, 1.00, 0.00, 1.00 };
// PushMatrix, GL_TEXTURE_2D, GL_TEXTURE_BINDING_2D, GL_CLAMP_TO_EDGE
// PushMatrix, GL_TEXTURE_2D, GL_TEXTURE_BINDING_2D, GL_CLAMP_TO_EDGE
// PushMatrix, GL_TEXTURE_2D, GL_TEXTURE_BINDING_2D, GL_CLAMP_TO_EDGE
// PushMatrix, GL_TEXTURE_2D, GL_TEXTURE_BINDING_2D, GL_CLAMP_TO_EDGE
// PushMatrix, GL_TEXTURE_2D, GL_TEXTURE_BINDING_2D, GL_CLAMP_TO_EDGE

f = $\frac{end - start}{end - start}$

GLfloat color[] = { 0.70, 0.70, 0.80, 1.00 };
glFogf(GL_FOG_COLOR, color);
glFogf(GL_FOG_START, 0.84);
glFogf(GL_FOG_END, 2.35);
glFogf(GL_FOG_MODE, GL_LINEAR);



KAPITAL LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA FUNDUSZ SPÓŁECZNY

Image Processing & Computer Graphics 49

Applications in biomedicine

Texturing Mesh rendering Faces rendering

KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA
FUNDUSZ SPOŁECZNY

Image Processing & Computer Graphics 50

Hardware graphics accelerator

A video adapter that contains its own processor that performs specialized set of computations (graphics functions) to boost performance of image rendering.

In acceleration a graphics pipeline is used. Dedicated hardware units are responsible for rendering stages required to transform for a 3D description to a 2D screen.

Some of the units are also duplicated to allow for parallel computation.

A shaders are procedures to calculate rendering effects on graphics hardware. One shader can be run on number of processing units in parallel to improve rendering efficiency.

KAPITAŁ LUDZKI
Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
UNIA EUROPEJSKA
FUNDUSZ SPOŁECZNY

Graphics processing unit - Wikipedia, the free encyclopedia - Windows Internet Explorer

http://en.wikipedia.org/wiki/Graphics_acceleration

article | discussion | edit this page | history

Graphics processing unit

From Wikipedia, the free encyclopedia
(Redirected from Graphics acceleration)

"GPU" redirects here. For other uses, see GPU (disambiguation).

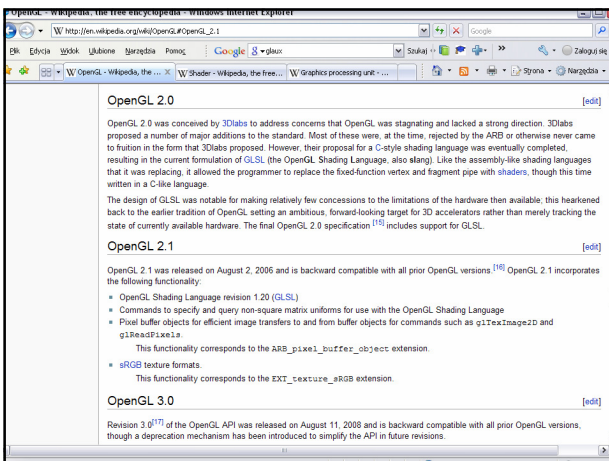
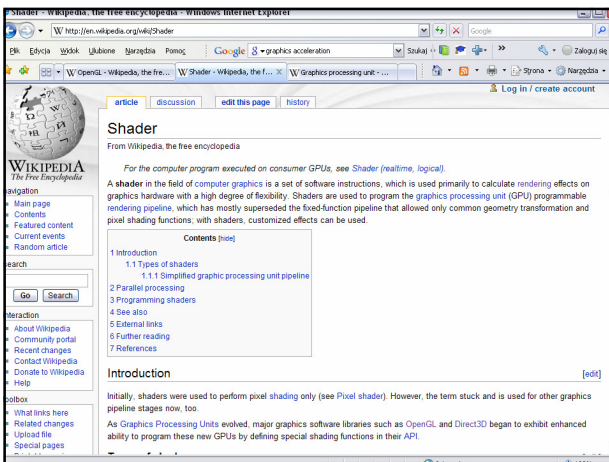
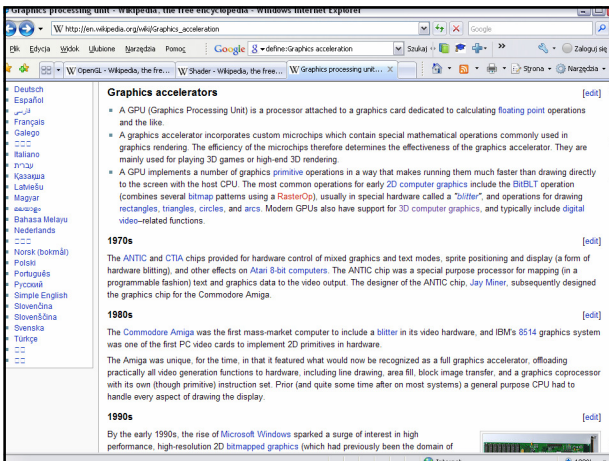
A graphics processing unit or GPU (also occasionally called visual processing unit or VPU) is a dedicated graphics rendering device for a personal computer, workstation, or game console. Modern GPUs are very efficient at manipulating and displaying computer graphics, and their highly parallel structure makes them more effective than general-purpose CPUs for a range of complex algorithms. A GPU can sit on top of a video card, or it can be integrated directly into the motherboard. More than 95% of new desktop and notebook computers have integrated GPUs, which are usually far less powerful than those on a video card.^[1]

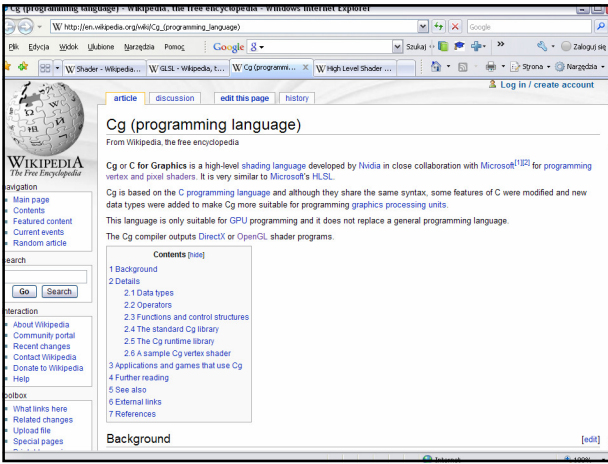
Contents [hide]

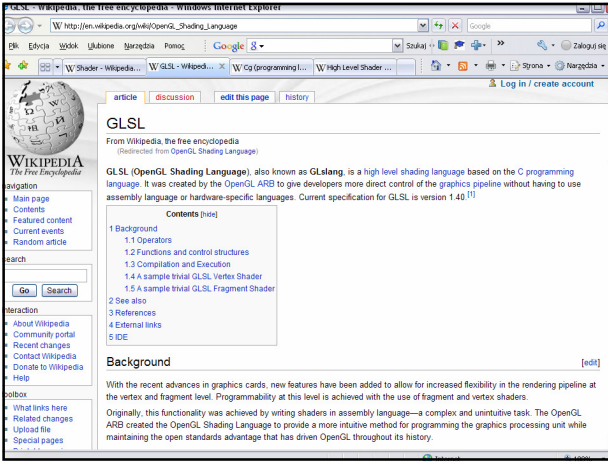
- 1 History
 - 1.1 1970s
 - 1.2 1980s
 - 1.3 1990s
 - 1.4 2000 to present
- 2 GPU companies
- 3 Computational functions
- 3 GPU forms
 - 3.1 Dedicated graphics cards
 - 3.2 Integrated graphics solutions
 - 3.3 Hybrid solutions
 - 3.4 Stream Processing and General Purpose GPUs (GPGPU)

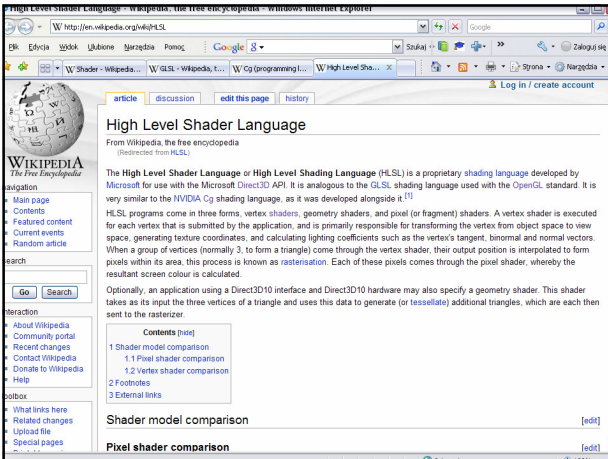
4 See also

GeForce 6800GT (NV43) GPU









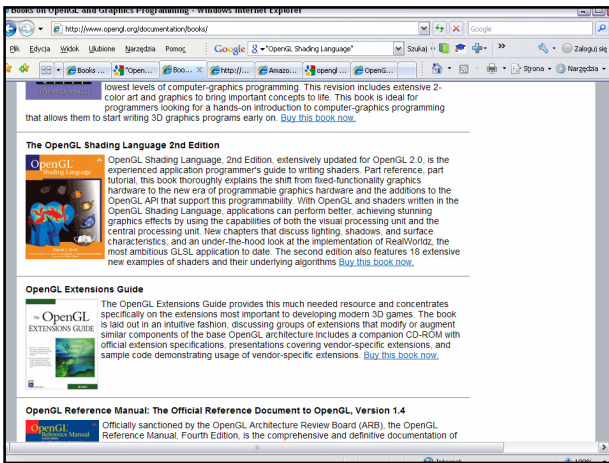


Image Processing & Computer Graphics
59

Thus, GPUs are powerful parallel processing/computation units.

...

Can we use GPUs for computations not related with 3D rendering?

Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego

From Wikipedia, the free encyclopedia

This article includes a list of references or external links, but its sources remain unclear because it lacks inline citations. Please improve this article by introducing more precise citations where appropriate. (November 2008)

Please help improve this article or section by expanding it. Further information might be found on the talk page. (August 2008)

General-purpose computing on graphics processing units (GPGPU, also referred to as GPGP and to a lesser extent GPP) is the technique of using a GPU, which typically handles computation only for computer graphics, to perform computation in applications traditionally handled by the CPU. It is made possible by the addition of programmable stages and higher precision arithmetic to the rendering pipelines, which allows software developers to use stream processing on non-graphics data.

Contents [hide]

- 1 GPU improvements
 - 1.1 Programmability
 - 1.2 Data types
- 2 GPGPU programming concepts
 - 2.1 Stream processing
 - 2.2 GPU programming concepts
 - 2.2.1 Computational resources
 - 2.2.2 Textures as stream
 - 2.2.3 kernels
 - 2.2.3.1 Flow control
 - 2.3 GPU techniques
 - 2.3.1 Map

The screenshot shows the Wikipedia article for 'CUDA'. The title is 'CUDA - Wikipedia, the free encyclopedia'. The article text explains that CUDA (originally an acronym for Compute Unified Device Architecture) is a parallel computing architecture developed by NVIDIA. It is used to program NVIDIA GPUs. The article mentions that CUDA works with all NVIDIA GPUs from the GBX series onwards. It also notes that the latest NVIDIA GPUs effectively become architectures like CPUs, but with a parallel 'many-core' architecture. A table on the right provides technical details:

CUDA	
Developed by	NVIDIA Corporation
Latest release	2.2 / April 2009
Operating system	Windows Vista, Windows XP, Windows Server 2008, Windows Server 2003, Linux, Mac OS X
Type	GPGPU
License	Proprietary, freeware
Website	Nvidia's CUDA zone

The screenshot shows the Wikipedia article for 'Close to Metal'. The title is 'Close to Metal - Wikipedia, the free encyclopedia'. The article text states that 'Close to Metal' (CTM, in short, originally called Close-to-the-Metal) is a name of a beta version of a low-level programming interface developed by ATI (now AMD Graphics Products Group). It aimed at enabling GPGPU computing. The article also mentions that the first production version of AMD's GPGPU technology is now called Stream SDK. A table of contents is provided:

Contents
1 Overview
2 Open Source
3 See also
4 References
5 External links

The 'Overview' section begins with: 'CTM gave developers direct access to the native instruction set and memory of the massively parallel computational elements in modern AMD video cards. CTM replaced the graphics-centric DirectX and OpenGL APIs for the GPGPU programmer while exposing previously unavailable low-level functionality. R590 (ATI x1900) and later generations of AMD's GPU microarchitectures supported the CTM interface.'

The slide is titled 'Summary, discussion and quiz' and is part of a presentation on 'Image Processing & Computer Graphics' (slide 63). At the bottom, there are logos for 'KAPITAŁ LUDZKI' and 'Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego'.



KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI



„ Image Processing and Computer Graphics”

Prezentacja multimedialna współfinansowana przez
Unię Europejską w ramach
Europejskiego Funduszu Społecznego w projekcie pt.
*„Innowacyjna dydaktyka bez ograniczeń - zintegrowany
rozwój Politechniki Łódzkiej - zarządzanie Uczelnią,
nowoczesna oferta edukacyjna i wzmacniania zdolności
do zatrudniania osób niepełnosprawnych”*



Politechnika Łódzka, ul. Żeromskiego 116, 90-924 Łódź, tel. (042) 631 28 83
www.kapitalludzki.p.lodz.pl
