ARTEFAKTY

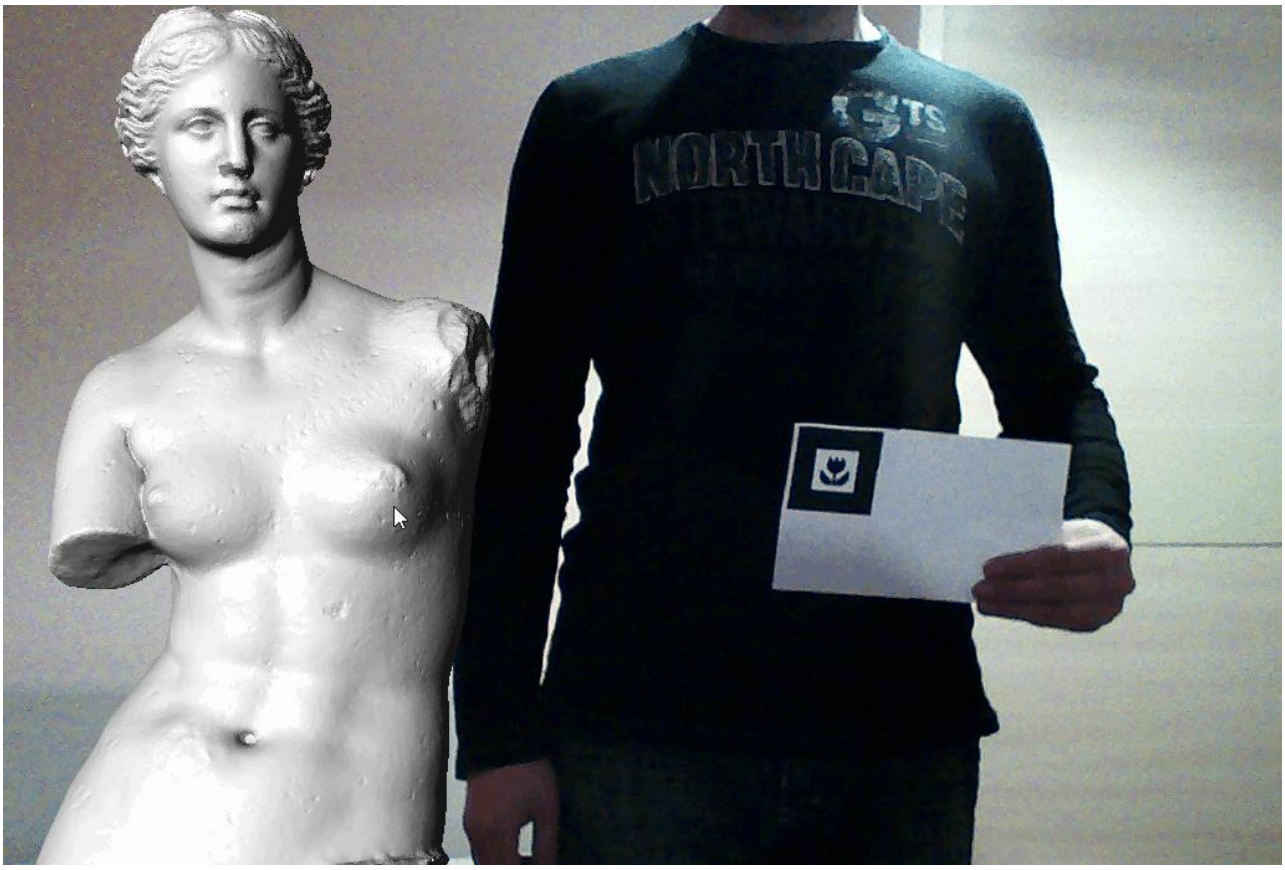
Aplikacja „Artefakty” wykorzystująca rozszerzoną rzeczywistość

Lublin 2016

SPIS TREŚCI

1. WSTĘP.

Aplikacja "Artefakty" oparta jest na ARToolKit. Jest to darmowa biblioteka oparta na licencji GNU General Public Lisense, stworzona do budowy aplikacji Rozszerzonej Rzeczywistości. Aplikacja nakłada wirtualny obraz modelu rzeźby Venus z Milo na obraz rzeczywisty. Program ten działa na systemie operacyjnym Windows XP, 7, 8, 10 32bit oraz 64bit . Do działania aplikacji wymagana jest kamera usb lub wbudowana. W programie użyto darmowego obiektu 3d, który należało dopasować oraz przekonwertować w programie MeshLab do pliku vrml. Poniżej zrzuty ekranu z aplikacji.



RYSUNEK 1.1 WYŚWIETLENIE OBIEKTU 3D NA PODSTAWIE MARKERA PO POWIĘKSZENIU I PRZESUNIĘCIU

RYSUNEK 1.2 WYŚWIETLENIE OBIEKTU NA PODSTAWIE MARKERA

2. PRZYGOTOWANIE I URUCHOMIENIE ŚRODOWISKA.

- Zainstaluj Visual Studio 2010 (obraz instalatora dołączony na płycie DVD).
- Skopiuj folder ARToolKit na dysk twardy komputera np. do folderu **Program Files**.
- Skopiuj biblioteki : **glu32.dll, glut32.dll opengl32.dll** z katalogu **OpenGL**, znajdującego się na płycie. Pozostaw stare, jeśli istnieją.

  Kopiuj odpowiednio do katalogu :
  - *C:\Windows\System32* - system operacyjny 32-bitowy
  - *C:\Windows\SysWOW64* -system operacyjny 64-bitowy
- Skopiuj zawartość folderów **lib** oraz **include**, znajdujących się w katalogu: **OpenGL\Visual Studio Net** do odpowiednich im folderów w lokalizacji: *C:\Program Files\Visual Studio 10.0\VC.*
- Otwórz plik **ARToolKit.sln** znajdujący się w głównym katalogu **ARToolKit**.

- Skompiluj projekt w trybie Debug. Podczas kompilacji nie powinny się pojawiać żadne poważne błędy.
- Otwórz plik **OpenVRML.sln**, który znajduje się w katalogu:
  C:\Program Files\ARToolKit\OpenVRML\src\openvrml-0.14.3\ide-projects\Windows\\VisualC7\OpenVRML.
- Zmień zawartość kilku plików:
  - **regerror.c** (regex -> Source Files)

    regerror(errcode, preg, errbuf, errbuf_size)
    int errcode;
    const regex_t *preg;
    char *errbuf;
    size_t errbuf_size;

    Zamień na:

    regerror(int errcode, regex_t *preg, char *errbuf, size_t errbuf_size)

  - **AST.hpp** (antlr -> Header Files)

```
inline operator<(RefAST l,RefAST r); // {return true;}
```

    Zamień na:

```
int inline operator<(RefAST l,RefAST r); // {return true;}
```

  - **Token.hpp** (antlr -> Header Files)

- Skompiluj projekt w trybie Release.
- Skopiuj biblioteki utworzone podczas kompilacji znajdujące się w lokalizacjach:

  - *C:\Program Files\ARToolKit\OpenVRML\src\openvrml-0.14.3\ide-projects\Windows\\VisualC7\OpenVRML\antlr\Release\antlr.lib*

  - *C:\Program Files\ARToolKit\OpenVRML\src\openvrml-0.14.3\ide-projects\Windows\\VisualC7\OpenVRML\openvrml\Release\openvrml.lib*

- *C:\Program Files\ARToolKit\OpenVRML\src\openvrml-0.14.3\ide-projects\Windows\\VisualC7\OpenVRML\openvrml-gl\Release\openvrml-gl.lib*

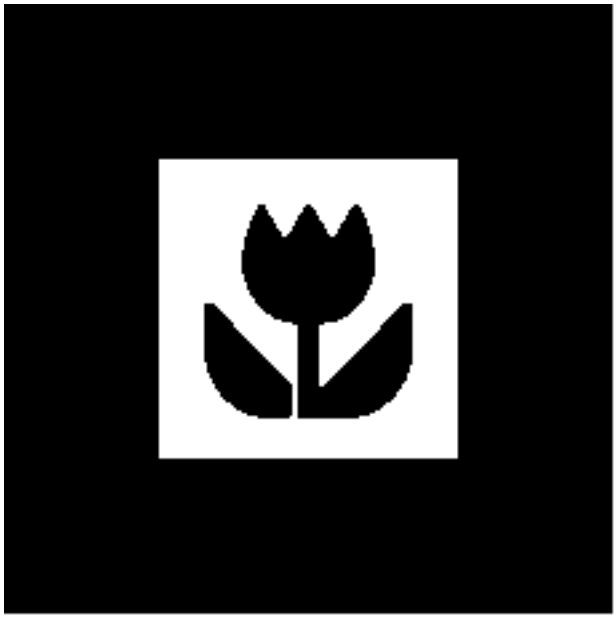
- *C:\Program Files\ARToolKit\OpenVRML\src\openvrml-0.14.3\ide-projects\Windows\\VisualC7\OpenVRML\regex\Release\regex.lib*

do folderu: *C:\Program Files\ARToolKit\OpenVRML\dependencies\lib*

- Otwórz ponownie **ARToolkit.sln**.
- W trybie Debug przejdź do właściwości klikając PPM na *Solution'ARToolkit'*.
  Następnie przejdź do opcji *Configure Properties*. W kolumnie *Build* zaznacz wszystkie możliwe pola wyboru.
- Przejdź do właściwości projektu *SimpleVRML*. Następnie przejdź do ustawień: *Linker -> General -> Output File* i zmień ustawienia na *<inherited from parent or project defaults>.*
- W trybie Release otwórz właściwości klikając PPM na *Solution'ARToolKit'*. Przejdź do opcji *Configure Properties*. W kolumnie *Build* zaznacz wszystkie możliwe pola wyboru.
- Rekompiluj ponownie cały projekt. Jeśli wystąpi dużo błędów należy przekompilować ponownie.
- W przypadku gdy po kompilacji wciąż mamy błędy należy skopiować lub zastąpić biblioteki znajdujące się w folderze: *C:\Program Files\ARToolKit\lib*, bibliotekami z katalogu **ARToolKitLib** z dołączonej płyty DVD.
- Sprawdź czy projekt się poprawnie skompilował tzn. czy istnieje plik **SimpleVRML.exe** w katalogu: *C:\Program Files\ARToolKit\bin.*
- Wydrukuj domyślny marker znajdujący się w katalogu: *C:\Program Files\ARToolKit\patterns\pattHiro.pdf.*
    - Uruchom **SimpleVRML.exe** i przystaw marker do kamery aby wyświetlić obiekt.

3. DODAWANIE WŁASNEGO MARKERA.

- Utwórz marker używając programu graficznego (Paint, Gimp) na podstawie pliku: **blankPatt.gif**. Jest to szablon markera, znajdujący się w katalogu: *C:\Program Files\ARToolKit\patterns.*
- Wydrukuj wcześniej utworzony marker - **marker.png** z katalogu: Marker znajdujący się na płycie DVD.
- Uruchom aplikację do rozpoznawania markerów: **mk_patt.exe** z lokalizacji: *C:\Program Files\ARToolKit\bin*. Należy ją uruchamiać z podniesionymi uprawnieniami (jako Administrator). Po uruchomieniu wpisz ścieżkę do domyślnej konfiguracji kamery: *"Data/camera_para.dat".*
- Ustaw marker w taki sposób, żeby krawędzie przy lewym górnym rogu podświetlały się na czerwono.
- W celu zapisania markera kliknij LPM na obrazie z kamery. Następnie przejdź do drugiego aktywnego okna i wpisz nazwę markera. Utworzony wcześniej marker znajduje się w katalogu: Marker pod nazwą **marker1.mo**. Należy go przekopiować do folderu: *C:\Program Files\ARToolKit\bin\Data.*



4. DODAWANIE WŁASNEGO OBIEKTU 3D.

1. Utwórz obiekt 3d za pomocą programu np. Blender lub MeshLab. Aby eksportować z Blendera obiekty zgodne z ARToolKit należy doinstalować dodatek Import-Export VRML2. Zmodyfikowany wcześniej obiekt znajduje się na płycie w katalogu Model3d pod nazwą: **venus_6.wrl**. Należy skopiować go do katalogu: *C:\Program Files\ARToolKit\bin\Wrl.* Model 3d pobrano z darmowej strony w formacie stl i przystosowano do zgodnego z Artoolkit.

RYSUNEK 4.1 HTTP://WWW.THINGIVERSE.COM/THING:196037

5. KONFIGURACJA POWIĄZANIA MARKERA Z MODELEM.

2. Utwórz plik z rozszerzeniem .dat. Ma on zawierać nazwę pliku modelu 3d oraz parametry związane z umiejscowieniem modelu względem markera tj. przesunięcie, obrót i skala.
   Przykładowy plik:

```
Venus_6.wrl
0.0 0.0 0.0             # Translation
0.0 0.0 0.0 0.0         # Rotation
10.0 10.0 10.0          # Scale
```

   W naszym przypadku jest to plik **model.dat**, znajdujący się w katalogu Model3d. Należy go skopiować do katalogu: *C:\Program Files\ARToolKit\bin\Wrl.*
3. Zmień zawartość pliku: **object_data_vrml** ustawiając w nim ścieżki dostępu do markerów oraz obiektów 3d. Kolejne parametry dotyczą wartości wysokości nałożonego obiektu oraz współrzędne w stosunku do środka markera.
   Przykładowy plik:

```
#the number of patterns to be recognized
1

#pattern 1
VRML    Wrl/model.dat
Data/marker1.mo
80.0
0.0 0.0
```

6. INSTALACJA APLIKACJI.

Aplikację instaluje się za pomocą instalatora Innosetup. Wystarczy uruchomić plik: Venusetup.exe znajdujący się na dołaczonej płycie. Instalator skopiuje niezbędne pliki i biblioteki potrzebne do uruchomienia aplikacji. Domyślnie aplikacja instaluje się w folderze: "Program Files\Artefakty". Tam też znajdują się zdjęcia markerów w plikach: marker.pdf lub marker.png. Należy je wydrukować i przystawiać do kamery podczas gdy program jest uruchomiony. Aby odinstalować program należy skorzystać z zakładki w Panelu Sterowania, służącej do odinstalowywania aplikacji. Poniżej skrypt do tworzenia instalatora w Innosetup.

```
; Script generated by the Inno Setup Script Wizard.
; SEE THE DOCUMENTATION FOR DETAILS ON CREATING INNO SETUP SCRIPT FILES!
#define MyAppName "Artefakty"
#define MyAppVersion "v.1.6
#define MyAppPublisher "MO"
#define MyAppURL "http://www.pollub.pl/"
#define MyAppExeName "venus.exe"
[Setup]
; NOTE: The value of AppId uniquely identifies this application.
; Do not use the same AppId value in installers for other applications.
; (To generate a new GUID, click Tools | Generate GUID inside the IDE.)
AppId={{5ECEF38E-7FE1-4B8A-8BAB-D5CABB45818E}
AppName={#MyAppName}
AppVersion={#MyAppVersion}
AppVerName={#MyAppName} {#MyAppVersion}
AppPublisher={#MyAppPublisher}
AppPublisherURL={#MyAppURL}
AppSupportURL={#MyAppURL}
AppUpdatesURL={#MyAppURL}
DefaultDirName={pf}\{#MyAppName}
DefaultGroupName={#MyAppName}
LicenseFile=C:\Pollub\aplikacja3\licencja.txt
InfoAfterFile=C:\Pollub\aplikacja3\after.rtf
OutputBaseFilename=Venusetup
Compression=lzma
SolidCompression=yes

[Languages]
Name: "english"; MessagesFile: "compiler:Default.isl"
Name: "polish"; MessagesFile: "compiler:Languages\Polish.isl"
[Tasks]
Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}";
GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked
```

```
[Files]
Source: "C:\Pollub\aplikacja3\venus.exe"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Pollub\aplikacja3\DSVL.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Pollub\aplikacja3\libARvideo.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Pollub\aplikacja3\licencja.txt"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Pollub\aplikacja3\marker.png"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Pollub\aplikacja3\marker.pdf"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Pollub\aplikacja3\msvcp71.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Pollub\aplikacja3\msvcr71.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Pollub\aplikacja3\msvcp100.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Pollub\aplikacja3\msvcr100.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Pollub\aplikacja3\glut32.dll"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Pollub\aplikacja3\vcredist_x64.exe"; DestDir: "{app}"; Check: IsWin64; Flags: ignoreversion
Source: "C:\Pollub\aplikacja3\vcredist_x86.exe"; DestDir: "{app}"; Check: "not IsWin64"; Flags: ignoreversion
Source: "C:\Pollub\aplikacja3\folder\*"; DestDir: "{app}"; Flags: ignoreversion recursesubdirs createallsubdirs
Source: "C:\Pollub\aplikacja3\glu32.dll"; DestDir: C:\WINDOWS\System32; Flags: onlyifdoesntexist uninsneveruninstall
Source: "C:\Pollub\aplikacja3\glut32.dll"; DestDir: C:\WINDOWS\System32; Flags: onlyifdoesntexist uninsneveruninstall
Source: "C:\Pollub\aplikacja3\opengl32.dll"; DestDir: C:\WINDOWS\System32; Flags: onlyifdoesntexist uninsneveruninstall
; NOTE: Don't use "Flags: ignoreversion" on any shared system files
[Icons]
Name: "{group}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"
Name: "{group}\{cm:UninstallProgram,{#MyAppName}}"; Filename: "{uninstallexe}"
Name: "{commondesktop}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"; Tasks: desktopicon
[Run]
Filename: "{app}\vcredist_x64.exe"; Parameters: "/passive /Q:a /c:""msiexec /qb /i vcredist.msi"" "; Check: IsWin64;
StatusMsg: Installing Visual Studio 2010 SP1 RunTime...
Filename: "{app}\vcredist_x86.exe"; Parameters: "/passive /Q:a /c:""msiexec /qb /i vcredist.msi"" "; Check: "not IsWin64";
StatusMsg: Installing Visual Studio 2010 SP1 RunTime...
Filename: "{app}\{#MyAppExeName}"; Description: "{cm:LaunchProgram,{#StringChange(MyAppName, '&', '&&')}}";
Flags: nowait postinstall skipifsilent
```

7. INSTRUKCJA OBSŁUGI PROGRAMU

- Wydrukuj marker znajdujący się w katalogu, gdzie został zainstalowany program np. : C:\Program Files\Artefakty. (marker.pdf lub marker.png).
- Uruchom aplikację "Artefakty" za pomocą skrótu na pulpicie lub z poziomu programów.
- Przystaw marker do kamery w taki sposób aby został rozpoznany przez program.
- Modyfikuj powstały obiekt za pomocą przycisków na klawiaturze:
  - "p" - włączenie opcji progowania, słóżącej do określenia jasności markera (wartość domyślna: 100, wartość max: 256).
  - "[" - zmniejszenie wartości progowania.
  - "]" - zwiększenie wartości progowania.
  - "-" - pomniejszenie obiektu 3d.
  - "+","=" - powiększenie obiektu 3d.
  - "w" - przesnięcie modelu w górę względem markera.

- "s" - przesnięcie modelu w dół względem markera.
- "a" - przesnięcie modelu w lewo względem markera.
- "d" - przesnięcie modelu w prawo względem markera.
- "strzałka w górę" ,"strzałka w dół" - obrót modelu względem osi x.
- "strzałka w lewo" ,"strzałka w prawo" - obrót modelu względem osi y.
- "," lub "<" , "." lub ">" - obrót modelu względem osi z.
- "c" - ustawienie rozdzielczości.
- "q","Esc"- wyłączenie programu.
-

8. KOD ŹRÓDŁOWY

PLIK "SIMPLEVRML.C":

```
#ifdef _WIN32
#  include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef __APPLE__
#  include <GLUT/glut.h>
#else
#  include <GL/glut.h>
#endif

#include <AR/config.h>
#include <AR/video.h>
#include <AR/param.h>
#include <AR/ar.h>
#include <AR/gsub_lite.h>
#include <AR/arvrml.h>

#include "object.h"



// ==========================================================================
```

```
#define VIEW_DISTANCE_MIN		10.0
#define VIEW_DISTANCE_MAX		10000.0


static int prefWindowed = TRUE;
static int prefWidth = 1280;
static int prefHeight = 720;
static int prefDepth = 32;
static int prefRefresh = 0;


static ARUint8		*gARTImage = NULL;


static int			gARTThreshhold = 100;
static long			gCallCountMarkerDetect = 0;
static int			scale = 50;
static GLfloat rotatex = 0.0;
static GLfloat rotatey = 0.0;
static GLfloat rotatez = 0.0;
static GLfloat translatex = 0.0;
static GLfloat translatey = 0.0;


static ARParam			gARTCparam;
static ARGL_CONTEXT_SETTINGS_REF gArglSettings = NULL;


static ObjectData_T	*gObjectData;
static int			gObjectDataCount;



static int setupCamera(const char *cparam_name, char *vconf, ARParam *cparam)
{
  ARParam			wparam;
	int				xsize, ysize;


  if (arVideoOpen(vconf) < 0) {
	fprintf(stderr, "setupCamera(): Unable to open connection to camera.\n");
	return (FALSE);
	}


  if (arVideoInqSize(&xsize, &ysize) < 0) return (FALSE);
  fprintf(stdout, "Camera image size (x,y) = (%d,%d)\n", xsize, ysize);

	// Load the camera parameters, resize for the window and init.
  if (arParamLoad(cparam_name, 1, &wparam) < 0) {
		fprintf(stderr, "setupCamera(): Error loading parameter file %s for camera.\n",
cparam_name);
```

```
    return (FALSE);
  }
  arParamChangeSize(&wparam, xsize, ysize, cparam);
  fprintf(stdout, "*** Camera Parameter ***\n");
  arParamDisp(cparam);

  arInitCparam(cparam);

        if (arVideoCapStart() != 0) {
        fprintf(stderr, "setupCamera(): Unable to begin camera data capture.\n");
                return (FALSE);
        }

        return (TRUE);
}

static int setupMarkersObjects(char *objectDataFilename, ObjectData_T **objectDataRef, int
*objectDataCountRef)
{

  if ((*objectDataRef = read_VRMLdata(objectDataFilename, objectDataCountRef)) == NULL) {
    fprintf(stderr, "setupMarkersObjects(): read_VRMLdata returned error !!\n");
    return (FALSE);
  }
  printf("Object count = %d\n", *objectDataCountRef);

        return (TRUE);
}

static void debugReportMode(const ARGL_CONTEXT_SETTINGS_REF arglContextSettings)
{
        if (arFittingMode == AR_FITTING_TO_INPUT) {
                fprintf(stderr, "FittingMode (Z): INPUT IMAGE\n");
        } else {
                fprintf(stderr, "FittingMode (Z): COMPENSATED IMAGE\n");
        }

        if (arImageProcMode == AR_IMAGE_PROC_IN_FULL) {
                fprintf(stderr, "ProcMode (X)   : FULL IMAGE\n");
        } else {
                fprintf(stderr, "ProcMode (X)   : HALF IMAGE\n");
        }

        if (arglDrawModeGet(arglContextSettings) == AR_DRAW_BY_GL_DRAW_PIXELS) {
                fprintf(stderr, "DrawMode (C)   : GL_DRAW_PIXELS\n");
        } else if (arglTexmapModeGet(arglContextSettings) == AR_DRAW_TEXTURE_FULL_IMAGE) {
                fprintf(stderr, "DrawMode (C)   : TEXTURE MAPPING (FULL RESOLUTION)\n");
        } else {
                fprintf(stderr, "DrawMode (C)   : TEXTURE MAPPING (HALF RESOLUTION)\n");
        }

        if (arTemplateMatchingMode == AR_TEMPLATE_MATCHING_COLOR) {
                fprintf(stderr, "TemplateMatchingMode (M)   : Color Template\n");
        } else {
```

```
		fprintf(stderr, "TemplateMatchingMode (M)   : BW Template\n");
	}

	if (arMatchingPCAMode == AR_MATCHING_WITHOUT_PCA) {
		fprintf(stderr, "MatchingPCAMode (P)   : Without PCA\n");
	} else {
		fprintf(stderr, "MatchingPCAMode (P)   : With PCA\n");
	}
}

static void cleanup(void)
{
	arglCleanup(gArglSettings);
	arVideoCapStop();
	arVideoClose();
#ifdef _WIN32
	CoUninitialize();
#endif
}

static void Keyboard(unsigned char key, int x, int y)
{
	int mode, threshChange = 0;
	int sk = 0;


	switch (key) {
		case 0x1B:
		case 'Q':
		case 'q':
			cleanup();
			exit(0);
			break;
		case 'C':
		case 'c':
			mode = arglDrawModeGet(gArglSettings);
			if (mode == AR_DRAW_BY_GL_DRAW_PIXELS) {
				arglDrawModeSet(gArglSettings, AR_DRAW_BY_TEXTURE_MAPPING);
				arglTexmapModeSet(gArglSettings,
AR_DRAW_TEXTURE_FULL_IMAGE);
			} else {
				mode = arglTexmapModeGet(gArglSettings);
				if (mode == AR_DRAW_TEXTURE_FULL_IMAGE)
	arglTexmapModeSet(gArglSettings, AR_DRAW_TEXTURE_HALF_IMAGE);
				else arglDrawModeSet(gArglSettings,
AR_DRAW_BY_GL_DRAW_PIXELS);
			}
			fprintf(stderr, "kamera - %f (frame/sec)\n",
(double)gCallCountMarkerDetect/arUtilTimer());
			gCallCountMarkerDetect = 0;
			arUtilTimerReset();
			debugReportMode(gArglSettings);
			break;
		case '[':
```

```
                threshChange = -1;
                break;
            case ']':
                threshChange = +1;
                break;
            case 'a':
                translatex -= 2;
                break;
            case 'd':
                translatex += 2;
                break;
            case 's':
                translatey -= 2;
                break;
            case 'w':
                translatey += 2;
                break;
                case '-':
                sk = -1;
                break;
            case '+':
            case '=':
                sk = +1;
                break;
            case ',':
            case '<':
                rotatez -=1;
                break;
            case '.':
            case '>':
                rotatez +=1;
                break;

            case 'P':
            case 'p':
                arDebug = !arDebug;
                break;
            default:
                break;
        }
        if (threshChange) {
            gARTThreshhold += threshChange;
            if (gARTThreshhold < 0) gARTThreshhold = 0;
            if (gARTThreshhold > 255) gARTThreshhold = 255;
            printf("poziom progowania %d.\n", gARTThreshhold);
        }
        if (sk) {
            scale += sk;
            if (scale < 1) scale = 1;
            if (scale > 500) scale = 500;
            printf("skala zmieniona na %d.\n", scale);
        }

}
```

```
static void SpecialKeys( int key, int x, int y )
{
  switch( key )
  {
  case GLUT_KEY_LEFT:
    rotatey -= 1;
    break;

  case GLUT_KEY_UP:
    rotatex -= 1;
    break;

  case GLUT_KEY_RIGHT:
    rotatey += 1;
    break;

  case GLUT_KEY_DOWN:
    rotatex += 1;
    break;
  }
}


static void mainLoop(void)
{
        static int ms_prev;
        int ms;
        float s_elapsed;
        ARUint8 *image;

        ARMarkerInfo    *marker_info;
  int       marker_num;
  int       i, j, k;


        ms = glutGet(GLUT_ELAPSED_TIME);
        s_elapsed = (float)(ms - ms_prev) * 0.001;
        if (s_elapsed < 0.01f) return;
        ms_prev = ms;


        arVrmlTimerUpdate();


        if ((image = arVideoGetImage()) != NULL) {
                gARTImage = image;

                gCallCountMarkerDetect++;


                if (arDetectMarker(gARTImage, gARTThreshhold, &marker_info, &marker_num) < 0) {
                        exit(-1);
                }
```

```
                for (i = 0; i < gObjectDataCount; i++) {


                        k = -1;
                        for (j = 0; j < marker_num; j++) {
                                if (marker_info[j].id == gObjectData[i].id) {
                                        if( k == -1 ) k = j; // First marker detected.
                                        else if (marker_info[k].cf < marker_info[j].cf) k = j;
                                }
                        }

                        if (k != -1) {

                                if (gObjectData[i].visible == 0) {
                                        arGetTransMat(&marker_info[k],
                                         gObjectData[i].marker_center,
                                        gObjectData[i].marker_width,
                                          gObjectData[i].trans);
                                        } else {
                                        arGetTransMatCont(&marker_info[k],
                                         gObjectData[i].trans,
                                         gObjectData[i].marker_center,
                                        gObjectData[i].marker_width,
                                        gObjectData[i].trans);
                                }
                                gObjectData[i].visible = 1;
                        } else {
                                gObjectData[i].visible = 0;
                        }
                }


                glutPostRedisplay();
        }
}

static void Visibility(int visible)
{
        if (visible == GLUT_VISIBLE) {
                glutIdleFunc(mainLoop);
        } else {
                glutIdleFunc(NULL);
        }
}

static void Reshape(int w, int h)
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glViewport(0, 0, (GLsizei) w, (GLsizei) h);

        glMatrixMode(GL_PROJECTION);
```

```
	glLoadIdentity();
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();

	}

static void Display(void)
{
	int i;
   GLdouble p[16];
	GLdouble m[16];

	glDrawBuffer(GL_BACK);
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear the buffers for new
frame.

	arglDispImage(gARTImage, &gARTCparam, 1.0, gArglSettings);
	arVideoCapNext();
	gARTImage = NULL;


	arglCameraFrustumRH(&gARTCparam, VIEW_DISTANCE_MIN, VIEW_DISTANCE_MAX, p);
	glMatrixMode(GL_PROJECTION);
	glLoadMatrixd(p);
	glMatrixMode(GL_MODELVIEW);


	glLoadIdentity();




	for (i = 0; i < gObjectDataCount; i++) {

		if ((gObjectData[i].visible != 0) && (gObjectData[i].vrml_id >= 0)) {


			arglCameraViewRH(gObjectData[i].trans, m, 2.0);
			glLoadMatrixd(m);


			glScalef( scale*0.01, scale*0.01, scale*0.01 );		//skalowanie
			glTranslatef(translatex,translatey,0.0);			//przesuwanie
			glRotatef(rotatex, 1.0, 0, 0 );
	//obracanie
			glRotatef(rotatey, 0, 1.0, 0 );
			glRotatef(rotatez, 0, 0, 1.0 );


			arVrmlDraw(gObjectData[i].vrml_id);
		}
	}
```

```
	glutSwapBuffers();
}



int main(int argc, char** argv)
{
	int i;
	char glutGamemode[32];
	const char *cparam_name = "Data/camera_para.dat";

#ifdef _WIN32
	char			*vconf = "Data\\WDM_camera_flipV.xml";
#else
	char			*vconf = "";
#endif
	char objectDataFilename[] = "Data/object_data_vrml";


	glutInit(&argc, argv);


	if (!setupCamera(cparam_name, vconf, &gARTCparam)) {
		fprintf(stderr, "main(): Unable to set up AR camera.\n");
		exit(-1);
	}

#ifdef _WIN32
	CoInitialize(NULL);
#endif


	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
	if (!prefWindowed) {
		if (prefRefresh) sprintf(glutGamemode, "%ix%i:%i@%i", prefWidth, prefHeight,
prefDepth, prefRefresh);
		else sprintf(glutGamemode, "%ix%i:%i", prefWidth, prefHeight, prefDepth);
		glutGameModeString(glutGamemode);
		glutEnterGameMode();
	} else {
		glutInitWindowSize(prefWidth, prefHeight);
		glutCreateWindow(argv[0]);
	}

		if ((gArglSettings = arglSetupForCurrentContext()) == NULL) {
		fprintf(stderr, "main(): arglSetupForCurrentContext() returned error.\n");
		cleanup();
		exit(-1);
	}
	debugReportMode(gArglSettings);
	glEnable(GL_DEPTH_TEST);
	arUtilTimerReset();
```

```
        if (!setupMarkersObjects(objectDataFilename, &gObjectData, &gObjectDataCount)) {
                fprintf(stderr, "main(): Unable to set up AR objects and markers.\n");
                cleanup();
                exit(-1);
        }

          fprintf(stdout, "Pre-rendering the VRML objects...");
        fflush(stdout);
   glEnable(GL_TEXTURE_2D);
   for (i = 0; i < gObjectDataCount; i++) {
                arVrmlDraw(gObjectData[i].vrml_id);
   }
   glDisable(GL_TEXTURE_2D);
        fprintf(stdout, " done\n");

        glutDisplayFunc(Display);
        glutReshapeFunc(Reshape);
        glutVisibilityFunc(Visibility);
        glutKeyboardFunc(Keyboard);
        glutSpecialFunc(SpecialKeys);

        glutMainLoop();

        return (0);
}
```

PLIK OBJECT.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <AR/ar.h>
#include <AR/arvrml.h>
#include "object.h"

static char *get_buff(char *buf, int n, FILE *fp)
{
   char *ret;

   for(;;) {
      ret = fgets(buf, n, fp);
      if (ret == NULL) return(NULL);
      if (buf[0] != '\n' && buf[0] != '#') return(ret);    }
}

ObjectData_T *read_VRMLdata( char *name, int *objectnum )
{
   FILE        *fp;
   ObjectData_T  *object;
   char         buf[256], buf1[256];
```

```
int          i;

        printf("Opening model file %s\n", name);

if ((fp=fopen(name, "r")) == NULL) return(0);

get_buff(buf, 256, fp);
if (sscanf(buf, "%d", objectnum) != 1) {
                fclose(fp); return(0);
        }

        printf("About to load %d models.\n", *objectnum);

if ((object = (ObjectData_T *)malloc(sizeof(ObjectData_T) * (*objectnum))) == NULL) exit (-1);

for (i = 0; i < *objectnum; i++) {

   get_buff(buf, 256, fp);
   if (sscanf(buf, "%s %s", buf1, object[i].name) != 2) {
      fclose(fp); free(object); return(0);
   }

                printf("Model %d: %20s\n", i + 1, &(object[i].name[0]));

   if (strcmp(buf1, "VRML") == 0) {
      object[i].vrml_id = arVrmlLoadFile(object[i].name);
                        printf("VRML id - %d \n", object[i].vrml_id);
      if (object[i].vrml_id < 0) {
         fclose(fp); free(object); return(0);
      }
   } else {
                        object[i].vrml_id = -1;
                }
                object[i].vrml_id_orig = object[i].vrml_id;
                object[i].visible = 0;

   get_buff(buf, 256, fp);
   if (sscanf(buf, "%s", buf1) != 1) {
                        fclose(fp); free(object); return(0);
                }

   if ((object[i].id = arLoadPatt(buf1)) < 0) {
                        fclose(fp); free(object); return(0);
                }

   get_buff(buf, 256, fp);
   if (sscanf(buf, "%lf", &object[i].marker_width) != 1) {
                        fclose(fp); free(object); return(0);
                }

   get_buff(buf, 256, fp);
   if (sscanf(buf, "%lf %lf", &object[i].marker_center[0], &object[i].marker_center[1]) != 2) {
      fclose(fp); free(object); return(0);
   }
```

```
  }

  fclose(fp);

  return( object );
}
```

PLIK OBJECT.H

```
#ifndef __object_h__
#define __object_h__


#define  OBJECT_MAX     30

#ifdef __cplusplus
extern "C" {
#endif

typedef struct {
  char     name[256];
  int      id;
  int      visible;
  double   marker_coord[4][2];
  double   trans[3][4];
        int      vrml_id;
        int      vrml_id_orig;
  double   marker_width;
  double   marker_center[2];
} ObjectData_T;

ObjectData_T  *read_VRMLdata (char *name, int *objectnum);

#ifdef __cplusplus
}
#endif

#endif
```