Lecture 6  MFC Programming

Yong-Jin Liu
Raster graphics
Scan conversion of line

\( (x_1, y_1) \)

\( (x_2, y_2) \)

\( D_x \)

\( D_y \)
Scan conversion of line $\Rightarrow$

Closed polygon region
Region filling
OpenGL:
Draw point, line, face

```c
glBegin(parameter);
...
...
glEnd();
```

parameter: GL_POINTS, GL_LINES,
GL_POLYGON, GL_TRIANGLES
The first OpenGL program

Event and message driven mechanism in Windows operating system

Program run,

Enter waiting status,

“do nothing”
Event occurs (press mouse, press key on keyboard, resize the window size, etc)

Generate special event message

The System manages the event queue automatically (First come first serve)
Programmer writes event handler

Programmer registers events handler into system

Event happens

The system call event handler

Remove the event from event queue

Restore waiting status

Use GLUT library!
The first OpenGL program

First step: preprocessing

opengl32.lib glut32.lib glu32.lib

include head file <gl/glut.h> <gl/glu.h> <gl/gl.h>
The first OpenGL program

- main(): main function

```c
int _tmain(int argc, _TCHAR* argv[])
{
    return 0;
}
```

`glutInit(&argc, (char**) argv);`
//This funciton initializes toolkit,
//other parameters are about command line,
//do not use them here
int _tmain(int argc, _TCHAR* argv[])
{
    glutInit(&argc, (char**) argv);
    glutInitDisplayMode(GLUT_DEPTH |
                        GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(320, 320);
    glutCreateWindow("2012 Spring Course");
}
Depth buffer
int _tmain(int argc, _TCHAR* argv[])
{
    ...
    glutDisplayFunc(renderScene);
    //register redraw event handler into system
}
void renderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glBegin(GL_TRIANGLES);
        glVertex3f(-0.5, -0.5, 0.0);
        glVertex3f(0.5, 0.0, 0.0);
        glVertex3f(0.0, 0.5, 0.0);
    glEnd();
    glutSwapBuffers();
}
int _tmain(int argc, _TCHAR* argv[])
{
    ... ...  
    glutDisplayFunc(renderScene);
    // register redraw event handler into system
    glutMainLoop();
    // enters the GLUT event processing loop.
}
Example demo

First step:

Install and run Microsoft Visual Studio 2005 (Team Edition, Version 8.0)
Select File → New → Project
- Select Win32 and corresponding Win32 Console Application
- Input program name xxx, press OK, then a program with nothing is generated.
Add in the project:

- `opengl32.lib glut32.lib glu32.lib`
- include header files
  
  `<gl/glut.h>  <gl/glu.h>  <gl/gl.h>`
Click the right mouse button, select Properties
In Configuration → Linker → Input

Additional Dependencies:
- opengl32.lib
- glut32.lib
- glu32.lib
Add at the beginning of the program

```
#include<gl/glut.h>
#include<gl/glu.h>
#include<gl/gl.h>
```

A executable, simple OpenGL program!
Check default searching directory of the project
Check default searching directory of the project.
Start Page - Microsoft Visual Studio

Connect To Team Foundation Server

New Project...

Open Project...

Recent Projects

test
QtDirectX
temp
QtDirectX

Get Started
Guidance and Resources
Latest News

Welcome  Windows  Web  Cloud  Office  SharePoint

What’s New in Visual Studio 2010
Learn about the new features included:
Visual Studio 2010 Overview
What’s New in .NET Framework 4
What’s New in Visual C++
Customize the Visual Studio Start Page

Creating Applications with Visual Studio

Extending Visual Studio
New Project

Installed Templates
- Qt4 Projects
- Visual C++
  - ATL
  - CLR
  - General
  - MFC
  - Test
  - Win32
- Other Languages
- Other Project Types
- Database
- Modeling Projects
- Test Projects

Online Templates

.NET Framework 4
Sort by: Default

Type: Visual C++
A project for creating a Win32 console application

Name: <Enter_name>
Location: D:\
Solution name: <Enter_name>

Create directory for solution
Add to source control

OK Cancel
// test.cpp : Defines the entry point for the console application.

int main(int argc, char* argv[])
{
    return 0;
}
<table>
<thead>
<tr>
<th>Additional Dependencies</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ignore All Default Libraries</td>
<td></td>
</tr>
<tr>
<td>Ignore Specific Default Libraries</td>
<td></td>
</tr>
<tr>
<td>Module Definition File</td>
<td></td>
</tr>
<tr>
<td>Add Module to Assembly</td>
<td></td>
</tr>
<tr>
<td>Embed Managed Resource File</td>
<td></td>
</tr>
<tr>
<td>Force Symbol References</td>
<td></td>
</tr>
<tr>
<td>Delay Loaded Dlls</td>
<td></td>
</tr>
<tr>
<td>Assembly Link Resource</td>
<td></td>
</tr>
<tr>
<td>kernel32.lib;user32.lib;gdi32.lib;winspool.lib;comdlg32.lib;advapi32.lib;shlwapi.lib;bcrypt.lib;crypt32.lib;setupapi.lib;shell32.lib;ole32.lib;oleaut32.lib;uuid.lib</td>
<td></td>
</tr>
<tr>
<td>Include Directories</td>
<td>Library Directories</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>E:\Work-Pub\Courses\Programming\gl$includePath</td>
<td>E:\Work-Pub\Courses\Programming\lib$libraryPath</td>
</tr>
</tbody>
</table>

**Include Directories**
Path to use when searching for include files while building a VC++ project. Corresponds to environment variable INCLUDE.
```cpp
#include "stdafx.h"

int _tmain(int argc, wchar_t* argv[])
{
    return 0;
}
```
VC++ Directories editing in Tools > Options has been deprecated.

VC++ Directories are now available as a user property sheet that is added by default to all projects. Please click '?' for more information.
Installed Templates

- MFC ActiveX Control (Visual C++)
- MFC Application (Visual C++)
- MFC DLL (Visual C++)

Type: Visual C++

A project for creating an application that uses the Microsoft Foundation Class Library.

Name: <Enter_name>
Location: D:\
Solution name: <Enter_name>

- Create directory for solution
- Add to source control
Add at the beginning of the program:

```c
#include<gl/glut.h>
#include<gl/glu.h>
#include<gl/gl.h>
```

A executable, simple OpenGL program!
Add color control:

```c
void glColor3f(float red, float green, float blue);
```

Define at the beginning of the program

```c
float red = 1.0, blue = 1.0, green = 1.0;
```
float red=1.0, blue=1.0, green=1.0;

void renderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glColor3f(red, green, blue);
    glBegin(GL_TRIANGLES);
        glVertex3f(-0.5, -0.5, 0.0);
        glVertex3f(0.5, 0.0, 0.0);
        glVertex3f(0.0, 0.5, 0.0);
    glEnd();
    glutSwapBuffers();
}
int _tmain(int argc, _TCHAR* argv[])
{
    ...
    glutDisplayFunc(renderScene);
    //create menu (to control color)
    glutMainLoop();
    ...
}

int _tmain(int argc, _TCHAR* argv[])
{
    ...
    glutDisplayFunc(renderScene);
    glutCreateMenu(processMenuEvents);
    // register callback function processMenuEvents
    glutAddMenuEntry("Red", RED);
    // add option in menu
    glutAddMenuEntry("Blue", BLUE);
    glutAddMenuEntry("Green", GREEN);
    glutAddMenuEntry("White", WHITE);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    // create the connection to mouse button
    glutMainLoop();
}
void processMenuEvents(int option) {
    switch (option) {
        case RED : red = 1.0; green = 0.0; blue = 0.0; break;
        case GREEN: red = 0.0; green = 1.0; blue = 0.0; break;
        case BLUE : red = 0.0; green = 0.0; blue = 1.0; break;
        case WHITE : red = 1.0; green = 1.0; blue = 1.0; break;
    }
    glutPostRedisplay();
}
Add animation:

We can specify a function in `glutIdleFunc(function pointer parameter)` if there is no event to handle (i.e. event loop is in idle status), execute this function.
Add animation:

```cpp
int _tmain(int argc, _TCHAR* argv[])
{
    ...
    glutDisplayFunc(renderScene);
    glutCreateMenu(processMenuEvents);
    glutIdleFunc(renderScene);
    //no event happens, execute this function;
    glutMainLoop();
}
```
float angle = 0.0;

void renderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(angle, 0.0, 1.0, 0.0);
    glColor3f(red, green, blue);
    glBegin(GL_TRIANGLES);
        glVertex3f(-0.5, -0.5, 0.0);
        glVertex3f(0.5, 0.0, 0.0);
        glVertex3f(0.0, 0.5, 0.0);
    glEnd();
    angle += 0.1;
    glutSwapBuffers();
}
Selective course project 3

• Using OpenGL+GLUT
• Rotating triangle program
• In this week’s class
Example analysis

- How to store 2D shape in computer?
- A common method
  - Approximate smooth curve using off lines
  - Represent 2D shape using a set of off lines
- Other method
  - Smooth spline line, surface (next chapter)
Composed by 21 lines
Off line number in the file
Point number of the first line
First point of the first line
Second point of the first line

Point number of the second line
First point of the second line

...
Add at the beginning of the program

#include<gl/glut.h>

#include<gl/glu.h>

#include<gl/gl.h>

A executable, simple OpenGL program!
Example analysis

- How to store 2D shape in computer?
  - Approximate smooth curve using polylines
  - Represent 2D shape using a set of polylines
  - Store a set of polylines in file
  - Data format
Composed by 21 polylines
21  → Polyline number in the file
4  → Point number of the first polyline
169 118 → First point of the first polyline
174 120  → Second point of the first polyline
179 124
178 126
5
298 86  → Point number of the second polyline
304 92  → First point of the second polyline
310 104
314 119
29
32 435
...
Example Analysis

- How to store 2D shape in computer?
  - Data structure stored
  - How to read and display 2D shape?
  - Use standard C++ stream
  - Draw polylines on screen using GDI+
Multiple file I/O libraries

- First, the traditional **C/Unix-style** libraries, composed by a low level function, such as `read()` and `open();`
- Followed by the `<stdio.h>` of ANSI C library, includes `fopen()`, `fread()` and other functions.
- Some other library or framework with ownership, such as MFC, it has many of its own file processing class.
- These libraries are generally not cross-platform, and lack of type-safe support.
- Standard C++ provides an enhanced, object-oriented `<fstream>` library. `<fstream>` provides advanced auto control mechanism and robust.
CFileDialog dlg(TRUE, NULL, L"*.dat", OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT, L"*.dat", NULL);
if(dlg.DoModal() == IDOK) {
    fstream inStream;
inStream.open(dlg.GetPathName(), ios::in);
if(inStream.fail()) return;
inStream >> pl_num;
if (pl_num==0 || pl_num>10000) {
    AfxMessageBox(L"The input data file is wrong!"); return;
}
clean_shape();
shape = new polyline*[pl_num];
unsigned int i, j, c_pt_num;
for (j=0; j<pl_num; j++) { //read each polyline
    inStream >> c_pt_num;
    shape[j] = new polyline(c_pt_num);
    for (i=0; i<c_pt_num; i++)
        inStream >> shape[j]->pt_array[i].x >> shape[j]->pt_array[i].y;
}
inStream.close();
Invalidate();
}
GDI+ Programming in Windows System

- GDI (graphical device interface)
- GDI is the middle layer between application and hardware
- .net version of GDI is called GDI+
- More easy to use, has huge ability of image processing
Example Analysis

First step:
Install and run Microsoft Visual Studio 2005 (Team Edition, Version 8.0)
Select File → New → Project
Select MFC and corresponding MFC Application

Input program name GDI+, a single document application is generated after press OK
A project for creating an application that uses the Microsoft Foundation Class Library

Type: Visual C++
GDI drawing

find OnDraw function in CGDIView class, Add the following code, then GDI drawing can be used.

```cpp
void CGDIView::OnDraw(CDC* pDC)
{
    CGDIDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: add draw code for native data here

    // GDI code
    CPen newPen(PS_SOLID, 3, RGB(255, 0, 0));
    CPen* pOldPen = pDC->SelectObject(&newPen);
    pDC->MoveTo(20, 10);
    pDC->LineTo(200, 100);
    pDC->SelectObject(pOldPen);
}
```
void CGDIVView::OnDraw(CDC* pDC)
{
    CGDIDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: add draw code for native data here
    //GDI code
    CPen newPen( PS_SOLID, 3, RGB(255, 0, 0) );
    CPen* pOldPen = pDC->SelectObject( &newPen );
    pDC->MoveTo( 20, 10 );
    pDC->LineTo( 200, 100);
    pDC->SelectObject( pOldPen );
}
GDI+ drawing

- Initialization of GDI+
- Add the following code in stdafx.h

```cpp
#include "gdiplus.h"
```

- Select Project → Property
- Add gdiplus.lib in Linker → Input
Additional Dependencies

- gdiplus.lib

Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.
- In the application class (CGDIApp), add a member variable

  ULONG_PTR m_gdiplusToken;

- Add `ExitInstance` function in the application class, and add the following code:

```cpp
int CGDIApp::ExitInstance(void)
{
    Gdiplus::GdiplusShutdown(m_gdiplusToken);
    return CWinApp::ExitInstance();
}
```
In the InitInstance function of the application class, add GDI+ initialization code:

```cpp
Gdiplus::GdiplusStartupInput gdiplusStartupInput;
Gdiplus::GdiplusStartup(&m_gdiplusToken,
                        &gdiplusStartupInput, NULL);
```

... ...
CGDIApp theApp;

// CGDIApp initialization
BOOL CGDIApp::InitInstance()
{
    // InitCommonControlsEx() is required on Windows XP if an application
    // manifest specifies use of ComCtl32.dll version 6 or later to enable
    // visual styles. Otherwise, any window creation will fail.
    INITCOMMONCONTROLSEX InitCtrls;
    InitCtrls.dwSize = sizeof(InitCtrls);
    // Set this to include all the common control classes you want to use
    // in your application.
    InitCtrls.dwICC = ICC_WIN95_CLASSES;
    InitCommonControlsEx(&InitCtrls);

    CWinApp::InitInstance();
    Gdiplus::GdiplusStartupInput gdiplusStartupInput;
    Gdiplus::GdiplusStartup(&m_gdiplusToken, &gdiplusStartupInput, NULL);

    // Initialize OLE libraries
    if (!AfxOleInit())
    {
        return FALSE;
    }
}

Now the drawing code of GDI+ can be used in the OnDraw function of View class:
//GDI code
CPen newPen( PS_SOLID, 3, RGB(255, 0, 0) );
CPen* pOldPen = pDC->SelectObject( &newPen );
pDC->MoveTo( 20, 10 );
pDC->LineTo( 200, 100);
pDC->SelectObject( pOldPen );

//GDI+ code
using namespace Gdiplus; // use namespace
Graphics graphics( pDC->m_hDC );
Pen pen(Color::Red, 3);
graphics.DrawLine(&pen, 20, 10, 200, 100);
GDI is a middle layer between application and hardware

GDI is a mid/low API, you need to know a little about hardware(device) information

GDI+ is a high level API, you do not device completely

GDI+ is powerful, here are some examples:
using namespace Gdiplus; // use namespace
Graphics graphics( pDC->m_hDC );
Pen pen(Color(255, 0, 0, 255), 15);

pen.SetDashStyle(DashStyleDash);
graphics.DrawLine(&pen, 0, 50, 400, 150);

pen.SetDashStyle(DashStyleDot);
graphics.DrawLine(&pen, 0, 80, 400, 180);

pen.SetDashStyle(DashStyleDashDot);
graphics.DrawLine(&pen, 0, 110, 400, 210);
In the engineering applications, pre-defined style of the brush sometimes can not meet the actual demand, we can define new style using SetDashPattern function.

```cpp
using namespace Gdiplus; // use namespace

Graphics graphics( pDC->m_hDC );
REAL dashVals[4] = {
    2, // length of short stroke
    2, // interval
    15, // length of short stroke
    2}; // interval
Pen pen(Color(255, 0, 0, 0), 5);
pen.SetDashPattern(dashVals, 4);
graphics.DrawLine(&pen, 5, 20, 405, 200);
```
Pen class of GDI+ also provides `SetStartCap` and `SetEndCap` method to set the style of start point and end point of line.

```cpp
using namespace Gdiplus;
Graphics graphics( pDC->m_hDC );

Pen pen( Color( 255, 0, 0, 255 ), 15);
pen.SetStartCap(LineCapFlat);
pen.SetEndCap(LineCapSquare);
graphics.DrawLine(&pen, 50, 50, 250, 50);

pen.SetStartCap(LineCapRound);
pen.SetEndCap(LineCapRoundAnchor);
graphics.DrawLine(&pen, 50, 100, 250, 100);

pen.SetStartCap(LineCapDiamondAnchor);
pen.SetEndCap(LineCapArrowAnchor);
graphics.DrawLine(&pen, 50, 150, 250, 150);
```
- **Brush and gradient**
  - brush is used to specify the character of filling
  - GDI+ provide SolidBrush and HatchBrush class for the brush of filling color and hatch
    - Horizontal, vertical lines, on the diagonal and lower diagonal, cross line, cross line, solid diamond
  - GDI+ provides LinearGradientBrush and PathGradientBrush class to create line gradient brush and path gradient respectively.
Line gradient is a transition (gradient) using two colors in a rectangular area, the direction of transition can be a horizontal, vertical and diagonal.

Path gradient brush is used to fill a closed path in a gradient color. Path gradient is a central color gradient mode, which gradual change from the center of the path to around.
using namespace Gdiplus;
Graphics graphics( pDC->m_hDC );

GraphicsPath path; // construct a path
path.AddEllipse(50, 50, 200, 100);

// create a brush using the path
PathGradientBrush pthGrBrush(&path);

// set the color of the center of the path to blue
pthGrBrush.SetCenterColor(Color(255, 0, 0, 255));

// set the color around the path to be blue
Color colors[] = {Color(0, 0, 0, 255)};
INT count = 1;
pthGrBrush.SetSurroundColors(colors, &count);

graphics.FillRectangle(&pthGrBrush, 50, 50, 200, 100);

LinearGradientBrush linGrBrush(
Point(300, 50),
Point(500, 150),
Color(255, 255, 0, 0), // red
Color(255, 0, 0, 255)); // blue

graphics.FillRectangle(&linGrBrush, 300, 50, 200, 100);
The following code constructs a complicated region using two spline curve, then fills it

```cpp
using namespace Gdiplus;

 Graphics graphics( pDC->m_hDC );

 Pen pen(Color::Blue, 3);
 Point point1( 50, 200);
 Point point2(100, 150);
 Point point3(160, 180);
 Point point4(200, 200);
 Point point5(230, 150);
 Point point6(220, 50);
 Point point7(190, 70);
 Point point8(130, 220);

 Point curvePoints[8] = {point1, point2, point3, point4, point5, point6, point7, point8};
 Point* pcurvePoints = curvePoints;

 GraphicsPath path;
 path.AddClosedCurve(curvePoints, 8, 0.5);

 PathGradientBrush pthGrBrush(&path);
 pthGrBrush.SetCenterColor(Color(255, 0, 0, 255));
 Color colors[] = {Color(0, 0, 0, 255)};
 INT count = 1;
 pthGrBrush.SetSurroundColors(colors, &count);

 graphics.DrawClosedCurve(&pen, curvePoints, 8, 0.5);
 graphics.FillPath(&pthGrBrush, &path);
```
Get detailed usage of GDI+ through MSDN Online Help
Optional orientation of transmission housing (90° steps)
Selective Course Project

* Write a code to read and display the dino.dat

* Draw a complicated 2D shape pattern using line, arc and other basic geometry elements.