Test comparativo per valutazione prestazioni grafiche
librerie OpenGL a confronto con Graphics (.NET)

Studente: Elvis Ciotti

Prof: V.Maniezzo, M.Roffilli

Corso. Algoritmi e Sistemi di Elaborazione
C.d.L. Scienze dell'Informazione

7/4/08

Introduzione

Il progetto ha lo scopo di valutare le performance di un sistema di visualizzazione (zoom, traslazione, pan) di un oggetto (grafo) con un elevato numero di elementi geometrici (archi e nodi). Ambiente di progettazione: Ms Visual C# 2003.

A confronto:

1) Grapics (classe Ms) in controllo pictureBox.
2) OpenGL (controlli e wrapper da librerie *tao framework*) con e senza supporto VBO (Vertex Buffer Object, cioè scrittura in memoria video e accesso direttamente ad essa tramite GCPU ad ogni operazione di trasformazione).

Librerie utilizzate e struttura del progetto

Ambiente di sviluppo: Ms Visual C# 2003 .NET

Librerie: Per utilizzare la libreria OpenGL e un controllo del form dove disegnare sono state utilizzate alcune librerie del progetto ”The Tao Framework” per .NET 2.0

http://sourceforge.net/projects/taoframework/

Librerie scaricabili dall'indirizzo indicato (pacchetto versione 2.0.0RC2). Le librerie utilizzate in particolare in questo progetto di testing sono:

```
Tao.OpenGl 2.1.0.3
Tao.Platform.Windows 1.0.0.3
```

Per la renderizzazione del testo si sono utilizzate le librerie de progetto “FreeType”
http://www.freetype.org/
modificate e compilate in modalità compatibile alle librerie sopra dal progetto “Icarus”

http://icarus.sourceforge.net/

Librerie Win32 scaricabili all'indirizzo

http://icarus.svn.sourceforge.net/viewvc/icarus/Bin/Win32/

sono necessarie (per questo progetto):

```
ISE.FreeType.dll
Tao.Externals.dll
freetype6.dll
```

Guida all'uso delle librerie

Scaricare le librerie (o installarle per Tao framework, quindi reperlirle dalla cartella *bin* nella root di installazione del framework), quindi aggiungere al progetto (come “References”, Solution Explorer -> References -> Add references -> Browse) i files *Tao.OpenGl.dll, Tao.Platform.Windows.dll, ISE.FreeType.dll* .

Inserire quindi nei sorgenti

```
using Tao.OpenGl;
using Tao.Platform;
```

Per visualizzare il controllo importato, spuntare *simpleOpenGLcontrol* nella personalizzazione della toolbar strumenti, quindi inserire il controllo *simpleOpenGLcontrol*.

Importante: subito dopo la *InitializeComponent()* del costruttore della classe, inserire:

```
simpleOpenGlControl1.InitializeContexts();
```

I comandi e costanti OpenGl sono richiamabili dall'oggetto **Gl**.
Esempio openGL (inizializzazione e disegno di una linea 2D):

```
Gl.glClearColor(1.0f, 1.0f, 1.0f, 0.0f);
Gl.glMatrixMode(Gl.GL_PROJECTION);
Gl.glLoadIdentity();
Gl.glOrtho(0, 600, 0, 400, -1, 0); //viewport, finestra di 600x400
Gl.glClear(Gl.GL_COLOR_BUFFER_BIT);
Gl.glColor3f(1.0f, 0.0f, 0.0f); //colore rosso
Gl.glBegin(Gl.GL_LINES);
Gl.glVertex2i(10, 10); //inizio linea
Gl.glVertex2i(590, 490); //fine linea
Gl.glEnd();
Gl.glFlush();
simpleOpenGlControl1.Refresh();
```

Gli ultimi due comandi sono necessari entrambi per visualizzare il disegno nel controllo.

Esempio di disegno con tecnologia VBO (Vertex Buffer Object) di 100 linee parallele.
(Per dettagli su VBO leggere la prima la sezione successiva “NOTE sull'UTILIZZO DI VBO”).

Inizializzazione (startup)

```
if (!GlExtensionLoader.LoadExtension("GL_ARB_vertex_buffer_object"))
{ /* VBO non supportata dalla scheda video */ }
int[]  vertici = vertici = new int[100];
int  indCorr = 0;
/*  inserisco nel vettore le coordinate consecutive delle linee
  * ad ogni ciclo memorizzo le coordinate di una linea orizzontale */
for (int j = 0; j < 100; j++)
{
   vertici[indCorr++] = 10;
   vertici[indCorr++] = j;
   vertici[indCorr++] = 590;
   vertici[indCorr++] = j;
 }
int[] buffers = new int[2]; //vettore degli indici dei buffers
Gl.glDeleteBuffers(1, buffers); //cancella precedenti memorizzazioni nella scheda video
Gl.glGenBuffers(1, buffers); //genera buffer nella scheda video
//attivo buffer
Gl.glBindBuffer(Gl.GL_ELEMENT_ARRAY_BUFFER, buffers[0]);

int dataSize = (indCorr * sizeof(int));
// copio nel buffer (memoria video) tutte le coordinate.
// l'ultimo parametro indica la modalità di aggiornamento (nessuno in questo caso)
Gl.glBufferData(Gl.GL_ELEMENT_ARRAY_BUFFER, (IntPtr)dataSize, vertici,
Gl.GL_STATIC_DRAW);
```

disegno dalla memoria video

```
///inizializzazioni matrici OpenGL, come le prime 5 righe dell'esempio precedente
[..]
//scelgo il buffer da utilizzare, già riempito
Gl.glBindBuffer(Gl.GL_ARRAY_BUFFER, buffers[0]);
Gl.glEnable(Gl.GL_VERTEX_ARRAY);
Gl.glVertexPointer(2, Gl.GL_INT, 0, (IntPtr)0);
Gl.glDrawArrays(Gl.GL_LINES, 0, N_LINEE);
Gl.glDisable(Gl.GL_VERTEX_ARRAY);
simpleOpenGlControl1.Refresh();
```

Esempio renderizzazione font

inizializzazione

```
ISE.FTFont fondo;
int s = 19;
fondo = new ISE.FTFont("C:\\WINDOWS\\Fonts\\arial.ttf", out s);
fondo.ftRenderToTexture(12, 72/*DPI*/);
```

render testo

```
fondo.ftBeginFont();
Gl.glPushMatrix();
Gl.glColor3f(0f, 0f, 0f); //nero
Gl.glTranslatef(50, 50, 0); //coordinate di posizionamento
fondo.ftWrite("testo renderizzato");
fondo.ftEndFont();
Gl.glPopMatrix(); //ripristino matrice
Gl.glFlush();
simpleOpenGlControl1.Refresh();
```

La renderizzazione del testo è utilizzata per disegnare le etichette nella soluzione "grafo" con OpenGL (senza VBO).

Altri esempi

- Aprire e analizzare la soluzione "taotest", sono implementate funzioni di base per l'utilizzo delle librerie. (figura a lato)
- Aprire e analizzare la soluzione "grafo", dove sono messe a confronto le diverse modalità di disegno di un grafo.

NOTE sull'UTILIZZO DI VBO (Vertex Buffer Objects)

Nell'utilizzo con VBO, vengono sfruttata l'accelerazione video di GCPU e VRAM garantendo prestazioni più elevate di una normale comunicazione RAM <-> CPU e chiamata a primitive video.

Il funzionamento di VBO è il seguente:

- Si crea un buffer di nome **B** con *glGenBuffers(...)* e si attiva *glBindBuffer(...)*
- Si copiano in memoria le coordinate consecutive degli elementi nel buffer B con *glBufferData*(...).
- Per ridisegnare:
- caricare la matrice identità e trasformazioni, impostare colore *GlColor*
- associare il buffer B *GlBindBuffer([...] B)*,
- impostare il tipo di elemento che usa le coordinate (es: glEnable(Gl.GL_VERTEX_ARRAY))
- impostare quanti punti dare ad ogni primitiva (2 in questo caso di linee) e da che punto. Gl.glVertexPointer(2, Gl.GL_INT, 0, (IntPtr)0);
- flush buffer *glFlush()*
- Refresh controllo OpenGlControl1 del form

Non tutte le schede video supportano il sistema OpenGL VBO, o lo supportano con basse prestazioni, privilegiando altre tecnologie di rendering e buffering.

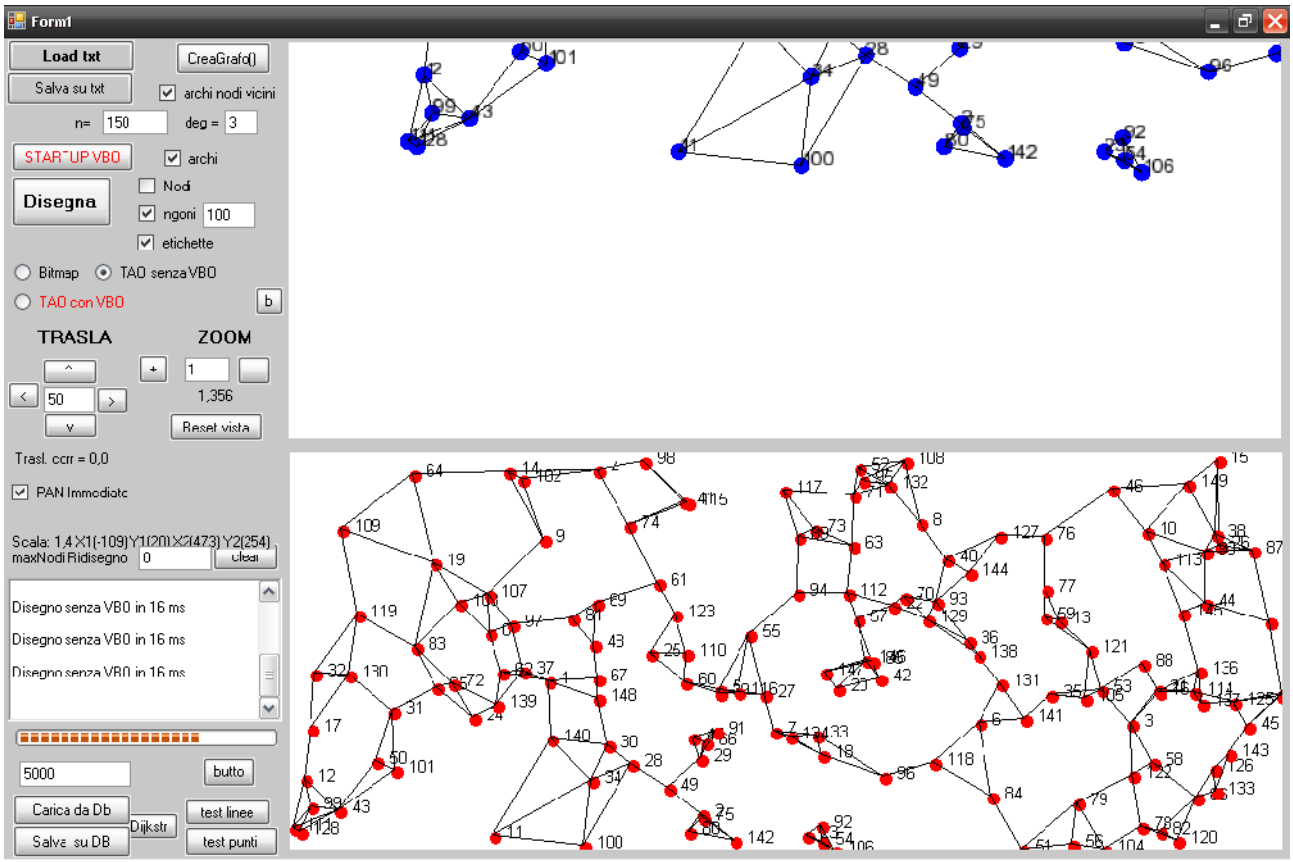
Preparazione e svolgimento dei test

Creazione grafo:
il grafo viene creato e tenuto in memoria con il numero di nodi N e grado M specificato, connettendo ogni nodo agli M nodi più vicini.

Il **disegno del grafo** avviene leggendo dalla memoria e chiamando le primitive di disegno, senza operazioni aggiuntive che compromettano la misurazione dei tempi.
Per ogni modalità si misurano le prestazioni con e senza disegno dei vertici ed etichette.
Modalità di disegno grafo

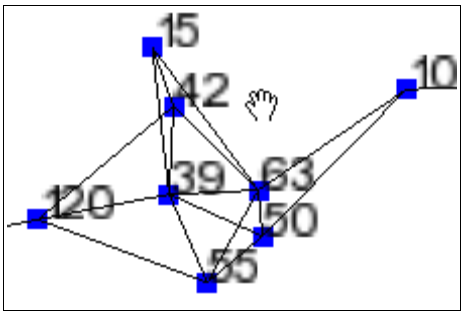
1) **PictureBox** (**Graphics**). Carico matrice identità, scala e traslazione. Per ogni nodo: DrawEllipse(), DrawString() e quindi M DrawLine() per gli archi fuoriuscenti dal nodo.
2) **OpenGL senza VBO**. Carico matrice identità, scala e traslazione. Gl.glColor3f, Gl.glBegin(Gl.GL_POINTS), quindi per ogni nodo:Gl.glVertex2i(), alla fine dei nodi.Gl.glEnd. Etichette: renderizzazione font con classe *FreeType*[2]). Per ogni arco fuoriuscente (nuovo ciclo sui nodi successivo): Gl.glBegin(Gl.GL_LINES); Gl.glVertex2i();Gl.glVertex2i(); [M volte]; Gl.glEnd(); Refresh controllo
3) **OpenGL con VBO.** In questo caso occorre una procedura iniziale di startup, nella quale si copiano in memoria video i nodi e vertici del grafo, per poi rileggerli molto più velocemente. (vedi sezione “NOTE sull'UTILIZZO DI VBO”)
   Procedura di **startup**: ciclo su nodi e vertici e copia coordinate in memoria video
   Procedura di **disegno**: Carico matrice identità, scala e traslazione corrente (impostata dalle operazione di traslazione, pan e zoom). Abilito lettura da memoria video Gl.glBindBuffer(), imposto colore rosso, leggo e disegno archi dalla memoria video (GCPU <-> VRAM, senza trasferimenti con RAM e utilizzo CPU), imposto colore nodi, leggo e disegno nodi dalla memoria video.

Breve guida all'utilizzo dell'ambiente di test (soluzione C#)



E' stato creata un'applicazione di gestione e testing per grafi. Di seguito brevi istruzioni all'uso.
Per creare un grafo e caricarlo in memoria: caricarlo da un file txt precedentemente salvato oppure cliccare su **creaGrafo()** specificando numero vertici e grado. Creare un grafo con "archi e nodi vicini" con molti nodi richiede tempi molto lunghi.
Cliccando su **Disegna**, verrà disegnato il grafo (correntemente in memoria) in una delle 3 modalità selezionate sotto con eventualmente nodi e etichette (caselle di spunta a lato).
Il disegno "TAO con VBO" richiede la procedura di **STARTUP**.
Nella textbox a sinistra il log delle operazioni e tempi di disegno.

NAVIGAZIONE / ESPLORAZIONE GRAFO

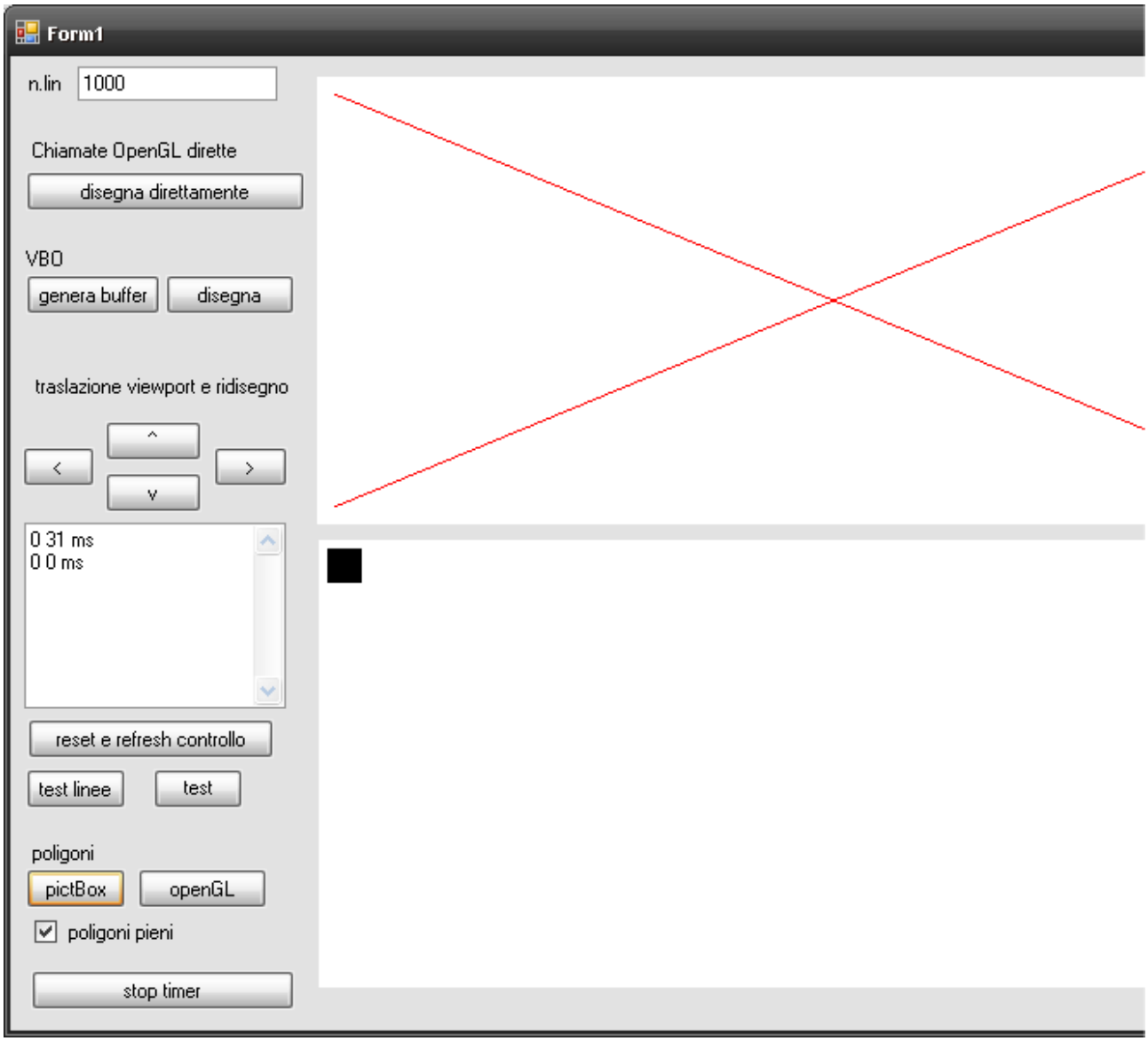


Se una delle 2 modalità OpenGL è selezionata (grafo disegnato in controllo in alto), è possibile esplorare il grafo con i comandi:
TRASLAZIONE: attraverso bottoni "TRASLA" con offset specificato
PAN: trascinando il mouse e rilasciandolo. Spuntare "PAN immediato" per effettuare un ridisegno continuamente (se la procedura di ridisegno richiede poco tempo, ad es. se la scheda supporta VBO).
ZOOM: usare i controlli in "ZOOM" o la **rotellina** del mouse (la vista verrà centrata sul puntatore)
Cliccare su "Reset Vista" per reimpostare zoom, traslazione di default.

ALTRI TEST

Utilizzare la soluzione "taotest" per testare le prestazioni di VBO nella propria configurazione con

il disegno di poligoni (e analizzare i sorgenti per capire il funzionamento).



Risultati dei test

Grafo 1: **100k** nodi - 188,7k archi (nodi vicini)
Grafo 2: **500k** nodi - 940k archi casuali

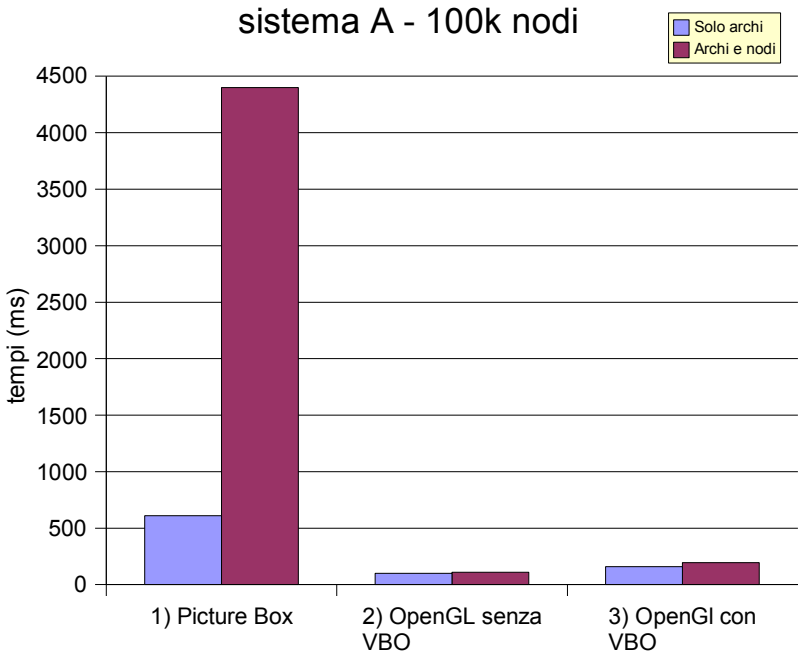
Risultati in millisecondi (libreria interna *DateTime*). Date le caratteristiche del framework, i risultati sulla misurazione dei tempi sono da considerarsi attendibili con una tolleranza di qualche decina di ms, si riportano quindi i range di valori misurati in più esecuzioni.
I test sono eseguiti ripulendo il codice funzioni overhead.
I conteggi **escludono** il caricamento delle matrici iniziali e inizializzazioni varie dei controlli. I conteggi includono funzioni di inizializzazione delle primitive grafiche (*glBegin GlEnd*), *glFlush*() e *refresh*() sul controllo.
Nel caso della misurazione con OpenGL, in alcune configurazioni i tempi stampati a video sono discordanti con i tempi effettivi per la visualizzazione (probabilmente dovuti a imprecisioni nei drivers / NET framework), in questi casi si sono misurati i tempi **effettivi** di visualizzazione.

AMBIENTE A (sistema portatile)

notebook HP dv 8000. CPU: Turion 64 (1600Mhz), RAM 512 Mb, SCHEDA VIDEO ATI mobility radeon Xpress 200M (processore grafico ATI X300 300/400Mhz, 128Mb – driver catalyst). La scheda video non supporta VBO, ma grazie al driver catalyst ne esegue tuttavia le istruzioni (con un evidente calo di prestazioni rispetto alla chiamata di primitive OpenGL da CPU e RAM.

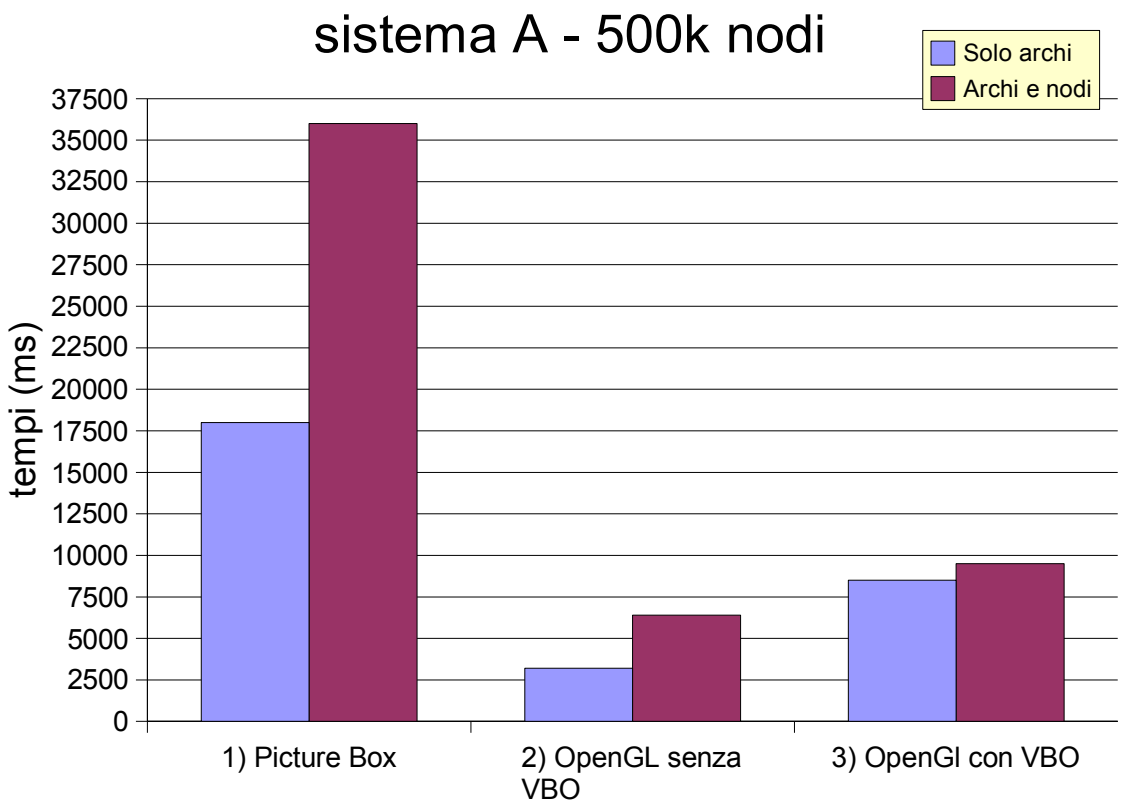
Grafo 1: **100k** nodi - 188,7k archi (nodi vicini)

	Solo archi	**Archi e nodi (points)**	Archi, nodi e etichette
1) Picture Box	609 - 625	4328 - 4453	6656 - 6703
2) OpenGL senza VBO	94 - 109	109-125	~9 sec*
3) OpenGl **con VBO** (Startup: ~160 ms)	156-171	187-203	n.d. **



Grafo 2: **500000** nodi - 940000 archi casuali

	Solo archi	**Archi e nodi**	Archi, nodi e etichette
1) Picture Box	~18 sec	~36 sec	~48 sec
2) OpenGL senza VBO	3234 - 3250	6406 - 3422	~35719 *
3) OpenGl **con VBO** (Startup: ~700 ms)	~8-9 sec***	9-10 sec***	n.d. **



* la renderizzazione è stata effettuata con libreria FreeType, con prestazioni che risultano ridotte.
** VBO per texture / font non supportato attualmente dalle librerie utilizzate.
*** valore calcolato manualmente, conteggio visibilmente errato da parte del framework .NET

CONSIDERAZIONI: OpenGL costituisce sicuramente un vantaggio, però la scheda gestisce male la modalità VBO con le primitive chiamate per questo grafo.

AMBIENTE B

CPU: Pentium D 3.4 Ghz, 3,37 Gb RAM, SCHEDA VIDEO Radeon **X300**

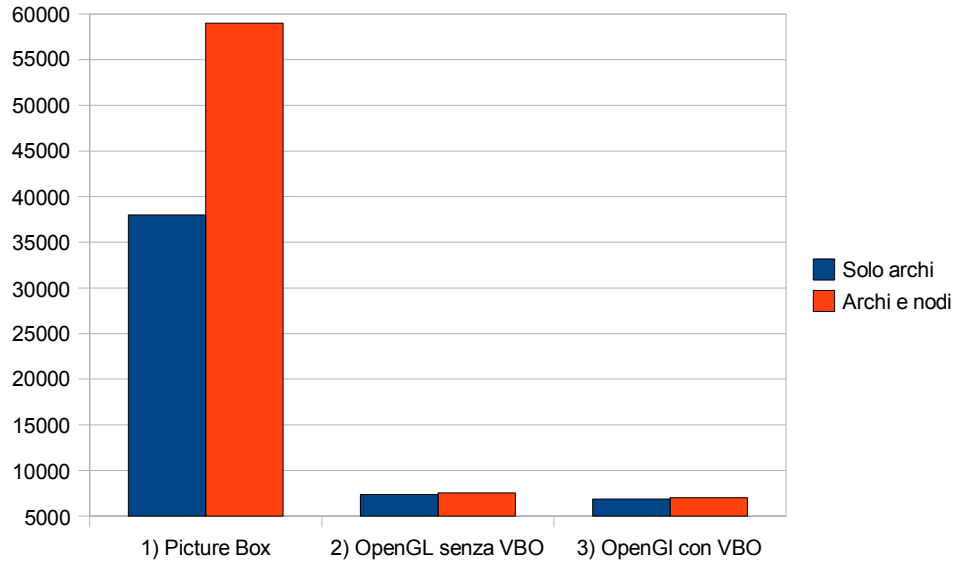
Grafo 1: **100000** nodi - 188700 archi (nodi vicini)

	Solo archi	**Archi e nodi**	Archi, nodi e etichette

1) Picture Box	672	4439	6189-6220
2) OpenGL senza VBO	31-47	31-47	2313-2329
3) OpenGl **con VBO** (Startup: ~60 ms)	390-406	500-515	nd

Grafo 2: **500000** nodi - **940000** archi casuali

	Solo archi	**Archi e nodi**	Archi, nodi e etichette
1) Picture Box	~38sec	~58sec	~67sec
2) OpenGL senza VBO	7346	7533	~19sec
3) OpenGl **con VBO** (Startup: ~ ms)	6850	6998	nd



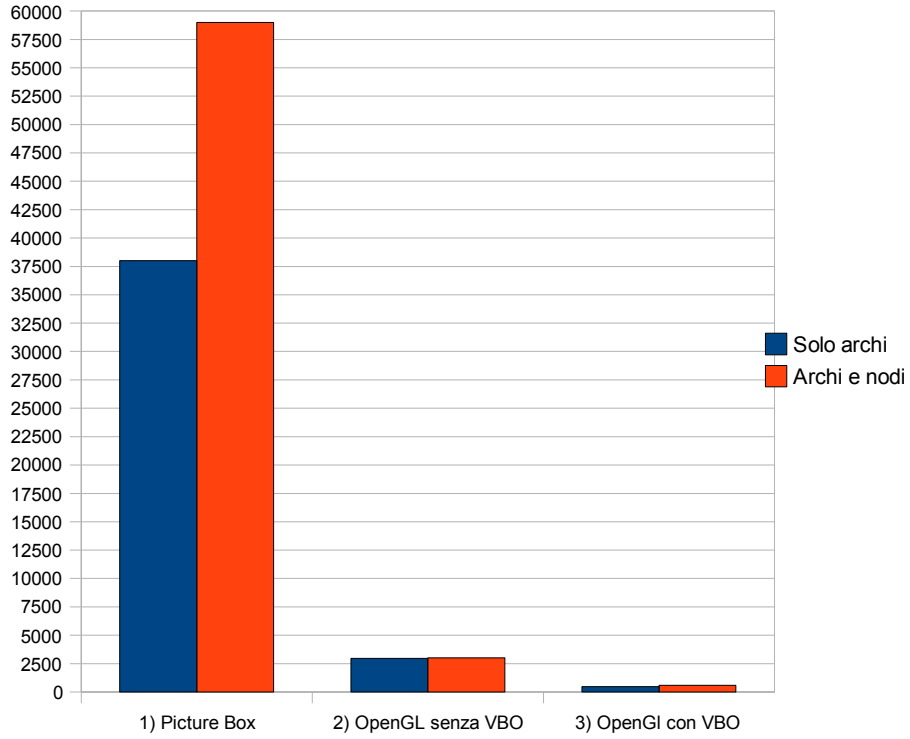
CONSIDeRAZIONI. L'ambiente B ha prestazioni che avvantaggiano VBO di poco con le primitive chiamate per questo grafo.

AMBIENTE C

CPU: pentium 4 3,4 Ghz, RAM 2 Gb, SCHEDA VIDEO ATI **radeon x600** pro

Grafo 2: **500000** nodi - **940000 archi casuali**

	Solo archi	**Archi e nodi**	Archi, nodi e etichette
1) Picture Box	~38 sec	~59 sec	~66 sec
2) OpenGL senza VBO	2953	3016	16172
3) OpenGl **con VBO** (Startup: ~ ms)	468	593	nd



CONSIDERAZIONI. L'incremento di prestazioni è notevole con OpenGL. VBO migliora ulteriormente i tempi.

TEST POLIGONI

Sulla soluzione “taotest” si è inserito un test di disegno di poligoni. Di seguito il codice nei due casi di disegno e la posizione delle istruzioni per la misurazione dei tempi.

Graphics:

```
//start conteggio
for (...)
g.FillPolygon(poligBrush, points);
// Pen arcpen = new Pen(Color.Black, 1); // se vuoti
//g.DrawPolygon(arcpen, points); // se vuoti
g.Dispose();
pictureBox_principale.Invalidate();
//end conteggio
```

OpenGL:

```
//start conteggio
for (...)
{
Gl.glBegin(Gl.GL_POLYGON);
// Gl.glBegin(Gl.GL_LINE_STRIP); //se vuoti
Gl.glVertex2i(points[0].X, points[0].Y);
Gl.glVertex2i(points[1].X, points[1].Y);
Gl.glVertex2i(points[2].X, points[2].Y);
Gl.glVertex2i(points[3].X, points[3].Y);
//Gl.glVertex2i(points[0].X, points[0].Y); //se vuoti (chiude poligono)
Gl.glEnd();
}
Gl.glFlush();
simpleOpenGlControl1.Refresh();
//end conteggio
```

Test effettuati sulla configurazione **A:**

test1) rettangoli (coprenti interamente la superficie di disegno): **590x250** pixel**.**

Tempi di disegno (ms)				
	Pol. Vuoto		Pol.Pieno	
Numero poligoni	**Graphics**	**OpenGL**	**Graphics**	**OpenGL**
10.000	453-468	31-46	4812	2843*
25.000	1171-1218	109	12203	6656*
50.000	2375	203	24656	12580
100.000	4750	515	49620	25625
500.000	24300	3203	248400	128456
1.000.000	nd	6656	nd	nd
5.000.000	nd	34000	nd	nd

* misurati con tasto “stop”. La chiamata “end conteggio” non ha dato valori coerenti con i risultati mostrati visivamente.

CONSIDERAZIONI:

I tempi crescono ovviamente in modo lineare con l'aumentare del numero di poligoni

Disegnando poligoni grandi come l'area di disegno (rettangoli **590x250):**

poligoni vuoti: Graphics impiega circa **7,58** volte il tempo di OpenGL

poligoni pieni: Graphics impiega circa **2** volte il tempo di OpenGL

test2) rettangoli : **20x20** pixel**.**

Tempi di disegno (ms)				
	Pol. Vuoto		Pol.Pieno	
Numero poligoni	**Graphics**	**OpenGL**	**Graphics**	**OpenGL**
10.000				
25.000	218-234	109	187-203	78-93
50.000				
100.000	890	437	781	437
500.000	4515	2203	3921	1781
1.000.000	9109	4406	7734	4375
5.000.000	45968	22140	39930	17906

* misurati con tasto “stop”. La chiamata “end conteggio” non ha dato valori coerenti con i risultati mostrati visivamente.

CONSIDERAZIONI: Disegnando poligoni grandi 20x20pixel**:**

poligoni vuoti: Graphics impiega circa **2** volte il tempo di OpenGL

poligoni pieni: Graphics impiega circa **2,2** volte il tempo di OpenGL

CONSIDERAZIONI PER ENTRAMBI I TEST

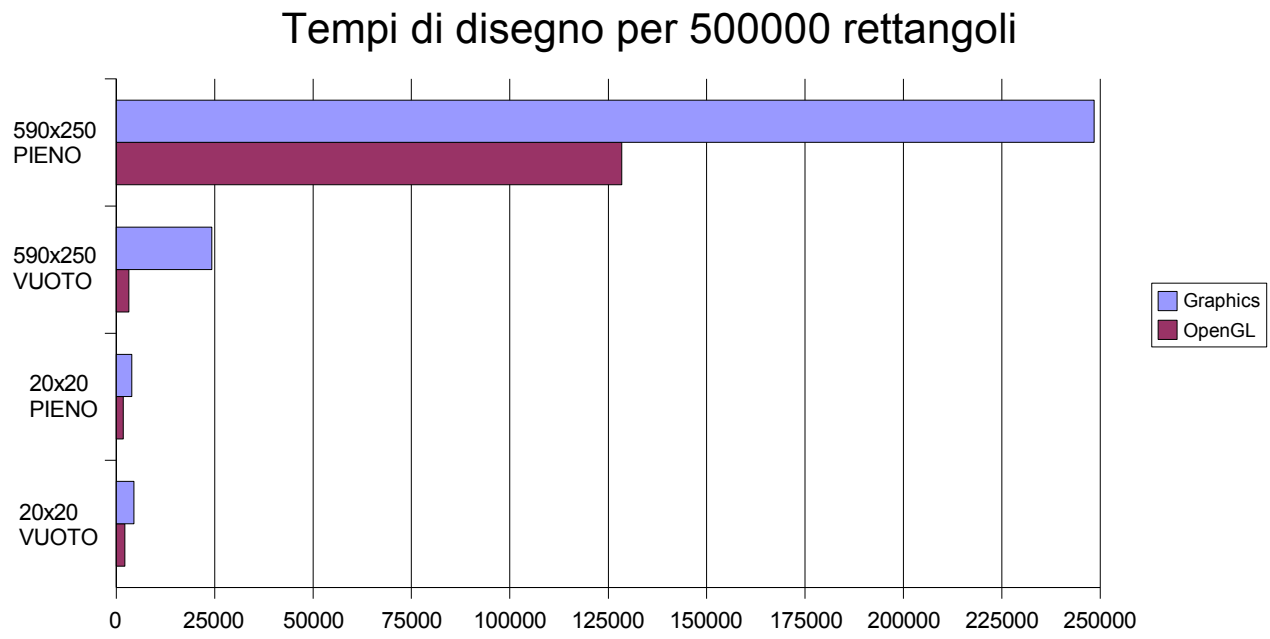
OpenGL ha sempre prestazioni migliori delle Graphics.

Per i rettangoli **pieni**, il rapporto tra le prestazioni Graphics / OpenGL è circa indipendente dalle dimensioni del rettangolo.

Per i rettangoli **vuoti**: Graphics ha scadenti prestazioni con rettangoli grandi. Impiega **7** volte i tempi di OpenGL per disegnare il rettangolo **grande**, mentre con rettangoli **piccoli** impiega solo **2** volte i tempi di OpenGL.

Questo indica che disegnare poligoni grandi vuoti con Grapics è sconveniente (l'istruzione ***DrawPolygon***(con *Pen* di dimensione 1) è **poco performante** in relazione agli altri risultati).

Nel grafico: il rapporti tra le coppie di colonne (Graphics / OpenGL) è sempre circa 2 tranne che nel secondo gruppo (cioè nel disegno di un rettangolo grande e vuoto).



TEST DISEGNO NODI GRAFO

Si sono testate le prestazioni di disegno di poligoni con n-lati (disposti a formare un cerchio), nella configurazione **A.** Inserendo la funzionalità nella soluzione "grafo" (generare il grafo e spuntare solo "ngoni" specificando numero di lati accanto).

Istruzioni utilizzate per le due modalità di disegno:

Per **Graphics**: *DrawEllipse()* di raggio 5.

Per **OpenGL**: *GlBegin*(GL_POLYGON) e quindi *GlVertex* nelle N (variabile) coordinate dei lati costituenti il perimetro di un cerchio di raggio 5.

Costruito un grafo di 500k nodi e disegnati SOLO i nodi:

	Tempi (ms)
Graphics - DrawEllipse()	19172 - 19312
OpenGL nodi seplici (points)	328
OpenGL 20-gono	9078 - 9382
OpenGL 50-gono	15188 - 15312
OpenGL 100-gono	25188 - 26906

CONSIDERAZIONI: I tempi di disegno di un vertice con OpenGL sono notevolmente inferiori a quelli per disegnare un cerchio DrawEllipse di Graphics. Nella configurazione A, le prestazioni di disegno di un cerchio con Graphics sono paragonabili al disegno di un n-gono con circa 70 lati e stesso raggio.

sorgenti

Mirroir per il download

http://www3.csr.unibo.it/studenti/elvis.ciotti/ase

http://www.elcisoft.com/ase