PRÁCTICA INTRODUCTORIA

Instalación de openGL y GLUT (o MESA y freeglut):

Todas las placas gráficas, que no sean de museo, tienen el soporte de OpenGL que vamos a necesitar, la versión 1.1 alcanza.
GLUT hay que bajarlo de Internet, sólo se necesitan los binarios y la documentación.
Los archivos de documentación de OpenGL GLU y GLUT están en la página de la cátedra, junto con el Red Book.

Windows:
De OpenGL no hay que instalar nada.
Si usan zinjai como IDE no tienen que bajar ni instalar nada extra.
Si usan Visual C++ o algún otro tendran que instalar glut o mejor freeglut:

glut
Bajen: http://www.xmission.com/~nate/glut/glut-3.7.6-bin.zip
En el zip hay tres archivos importantes:

- glut32.dll va en un directorio del path (system32).
- glut32.lib va en el directorio lib de la API de programación (ej: C:\Archivos de programa\Microsoft Visual Studio\VC98\Lib)
- glut.h va en el directorio include\GL (ej: C:\Archivos de programa\Microsoft Visual Studio\VC98\Include\GL)

freeglut:
Descarguen e instalen la última versión de: http://freeglut.sourceforge.net/.

Linux:
Como regla general, se deben instalar primero los drivers actualizados de la placa gráfica, normalmente ellos tienen el soporte de openGL por hardware. En caso contrario consultar.
Insatalar freeglut en versión de desarrollo (freeglut-dev o freeglut-devel).
El comando glxinfo debería dar la información de si el direct rendering (por hardware) esta activo o no.

Compilación:
Junto con el código fuente se ofrecen dos subdirectorios con los proyectos listos para usar desde Linux o zinjai o Visual Studio 6. Consultar para cualquier otra plataforma.

Código Fuente

Se empieza a leer desde la función main(). Ahí puede verse que el programa consiste de una etapa de inicialización y luego entra en el loop de ejecución, del cual nunca sale.
Inicialización:
Todo el trabajo se realiza en una rutina especial (excepto la inicialización interna de glut, para no pasarle los argumentos requeridos).
La inicialización de glut provee memoria y recursos. Se pide solamente lo que se va a utilizar.
Siempre que se pretenda trabajar en tiempo real se pide double buffer. En tal caso, mientras se ve el buffer "de adelante" (front), todo el dibujo se realiza en el buffer "de atrás" (back) y, cuando está listo, se intercambian (swap) los buffers, el de atrás pasa al frente y se actualiza la pantalla. Con la técnica de double buffering se evita el parpadeo (flickering) que se produce en las animaciones con un solo buffer, mientras se está dibujando sobre lo que se está viendo.
Respecto al color, siempre pedimos RGB (o RGBA, si vamos a usar transparencias, antialiasing o alguna forma de mezcla de colores). Los colores indexados o paletas (256) solamente se utilizan en casos muy especiales, todo lo que se solía hacer con una paleta limitada, se hace hoy con RGB estándar (3x8 bits).
En los próximos prácticos solicitaremos y utilizaremos algunos buffers más.
Los callbacks que se declaran corresponden a los eventos que queremos que el programa interprete. No tiene sentido hacer más lenta a la máquina por atender eventos que no vamos a procesar (movimiento del mouse, por ejemplo). Con este mismo sentido debemos cuidar la velocidad dentro de los callbacks: si no hay nada que hacer en él, hay que salir muy rápido.
El callback de display debe estar definido obligatoriamente y en realidad es necesario, al igual que el callback de reshape.
El resto de las inicializaciones de glut y las de openGL se corresponden con el estado inicial (ej.: tamaño inicial de ventana) o bien, para cosas que normalmente no se fijan en esta etapa (ej.: matrices), fijan el estado constante de funcionamiento.

Callbacks:
Se pueden leer siguiendo acciones, comencemos por el mas importante, el de dibujo.
En Display_cb() podemos ver como se arma la pantalla que se va a visualizar. Dado que todo lo dibujamos con openGL, casi todos los comandos de dibujo de openGL están allí o en rutinas que se llaman desde allí.
El redisplay no es implícito: cuando el programa modifica (edita) algo, debe llamar explícitamente al callback (o avisar de la necesidad con glutPostRedisplay); pero el sistema operativo también provoca llamadas implícitas; típicamente, al mover una ventana de otro programa por sobre la nuestra.
Descomentando la primera línea podemos ver, en la ventana de comandos, la impresionante cantidad de veces que se llama a esta rutina, por lo tanto es evidentemente un cuello de botella en el que hay que programar con mucho cuidado.
El redisplay explícito (llamando a Display_cb) se hace al terminar con una serie de modificaciones. Si en una rutina se desconoce si lo que acaba de modificarse es todo lo que hay que hacer (o se sabe que no es todo), en lugar de hacer la llamada explicita obligando a redibujar; se avisa, mediante una llamada a glutPostRedisplay(), que hay que hacerlo en el próximo paso por el loop de ejecución.
Respecto a los comandos de dibujo en sí, son la parte mas trivial de entender aquí, excepto por el significado de las coordenadas x e y.
Reshape_cb() se encarga de reaccionar a los cambios de tamaño de la ventana y también es llamado al crearse la ventana. En él se definen el viewport, que es la región de la ventana en donde se hará el dibujo, el sistema de coordenadas y la porción del espacio visible. En este caso (y casi siempre en los programas de dibujo trivial y 2D) las coordenadas miden simplemente píxeles en la ventana completa. La única dificultad es la falta de uniformidad en el origen: para los callbacks de glut y para el sistema operativo (cursor, posición de la ventana) las coordenadas son siempre píxeles, con el origen arriba a la izquierda; para OpenGL en cambio el origen está abajo a la izquierda y con z apuntando hacia fuera del monitor. Para solucionarlo en donde importa: sólo en las coordenadas del cursor, reemplazamos **y** por **h-y**.
Los detalles de las funciones de OpenGL que se usan aquí los veremos más adelante, al analizar las transformaciones y sus matrices.
La definición del sistema de coordenadas y la vista se hace donde haga falta, normalmente en una rutina especial, pero en este pequeño programa se hicieron en el callback de reshape porque es en el único momento en que cambian.
Para salir del programa se utiliza el estándar <ALT>+<F4> por lo que se declaró y definió el callback que atiende a las teclas especiales: Special_cb(). Como puede verse solo actúa si la tecla es <F4> y además el modificador <ALT> está pulsado. Los modificadores son <ALT>,<CTRL> o <SHIFT> o combinaciones ("OR-eadas"). Los detalles pueden verse en el manual de glut.
A partir de aquí seguimos cursos de acción más que callbacks.
Cuando el programa comienza no hay ningún punto definido; picando en la ventana se crean los puntos y cuando hay tres se visualiza el triángulo. Si se pica cerca de un punto previo, este puede moverse y también puede moverse el punto recién creado. Las teclas DEL y Backspace se programaron para borrar puntos. Como es estándar, una borra "hacia adelante" y la otra "hacia atrás" en la lista de puntos.
Las variables que almacenan los puntos y controlan la edición son globales y están definidas y explicadas al principio del archivo fuente. Solo resta decir que, por simplicidad, los puntos se almacenan en una lista unidimensional de pares de enteros {x0,y0,x1,y1,x2,y2,x3,y3}.
Con el funcionamiento del programa en mente, se puede seguir fácilmente la acción de cada evento, sólo hay que saber que callback maneja cada uno: Mouse_cb() se encarga los clicks, reacciona al pulsar y al soltar algún botón del mouse, en este caso sólo atendemos al botón izquierdo. Motion_cb() se activa cuando se mueve el cursor con algún botón apretado, esta acción se llama drag (arrastre), solo se define como callback cuando efectivamente se pulsó algún botón y se anula (se le pasa a glut la dirección nula) al soltarlo. El callback de los drags sigue al cursor aún fuera de la ventana, por lo que hay que programar si se permite o no el arrastre de un punto fuera de la ventana, puede suceder que quede invisible y por lo tanto no se puede volver a picar sobre el mismo para editarlo.
Por último, Keyboard_cb() atiende a las pulsaciones del teclado estándar (ASCII); en este caso solo "eschucha" DEL y Backspace.
Hay dos partes ligeramente complicadas: una consiste en averiguar si el botón izquierdo se pulsó con el cursor cerca de un punto previo y la otra es la lógica de borrar hacia delante o hacia atrás. Se espera que el alumno las entienda interactuando con el programa, utilizando breakpoints y watches o bien utilizando algunos cout a la ventana de comando.

Consignas

Probar, entender y cambiar cualquier cosa.
Programar que la tecla <esc> (ASCII 27) borre todo.
Programar para que no pueda sacarse un punto fuera de la ventana.
Modificar el programa para que el triangulo tenga líneas de borde.
Modificar para que haga un cuadrilátero.
¿Que está mal con el callback de teclado? (no "mal" sino cuestionable en términos de eficiencia)
Analizar la factibilidad (no es necesario programarlo) de agregar la siguiente funcionalidad: si el usuario pica dentro del triangulo (o del cuadrilátero), este se mueve completo.