Universidad Simón Bolívar
Departamento de Computación
y Tecnología de la Información
CI 4321 - Computación Gráfica I

Proyecto 1: Breakout

Integrantes:
Fernando Torre (05-38990)
Fernando Lovera (07-41126)

Contenido

Proyecto 1: Breakout 1
Contenido 2
Introducción 3
Ejecución 3
Organización del código 4
Estructuras definidas 5
Implementación de simulación física 8
Interacción con GLUT 9
Implementación de características fundamentales de *Breakout* 10
Implementación de especificaciones adicionales 11
Conclusiones 12

Introducción

Desde su invención en 1970, Breakout se ha mantenido como uno de los videojuegos más populares de grandes y jóvenes por igual. Breakout es el sucesor espiritual de Pong y consiste en usar una paleta o barra direccionadora para hacer rebotar una pelota y que la misma destruya una serie de ladrillos de colores en el campo de juego, sin que la pelota se salga del área de juego (una caja a la cual le falta una pared).

A pesar de que fue diseñado para computadoras analógicas con lógica discreta, Breakout ha sido muy popular en todo tipo de artefactos electrónicos desde televisores hasta celulares. El objetivo de este proyecto es replicar este éxito con la establecida Herramienta de Utilidad para Librería Gráfica Abierta (Open Graphics Library Utility Toolkit) o GLUT.

La ejecución del mismo es bastante sencilla. El código se encuentra organizado de una manera coherente la cual se explica en el presente, seguido de las estructuras definidas para el proyecto, así como enumerar el propósito y razonamiento detrás de cada una de las funciones esenciales.

Ejecución

El programa adjunto es compatible con cualquier sistema operativo.
Para ejecutar el programa, simplemente debe abrir una consola (Linux) o símbolo de sistema (Windows) y escribir el comando “make”. Este comando efectuará la compilación de todos los archivos, finalmente produciendo un ejecutable llamado “Breakout.exe”

Para llamar al programa, es necesario un archivo de configuración (el comportamiento del programa es no-determinístico sin él). Las llamadas son de la forma

```
$ ./Breakout.exe <archivo>
```

donde “$” es el “prompt” de la cónsola y `<archivo>` es el nombre del archivo de configuración. Así, si el nombre del archivo es “juego.txt”, la llamada sería “`./Breakout.exe juego.txt`”

Al inicio del juego, se carga el nivel 1 del archivo. El jugador puede mover la paleta con las flechas izquierda y derecha. La pelota se moverá junto con la paleta hasta que el jugador presione la barra espaciadora. La pelota es lanzada verticalmente con un ángulo de [±5]° escogido al azar.

Al impactar un ladrillo, este cambia de color, volviéndose progresivamente más amarillo hasta que se pueda destruir. La puntuación resultante de impactar un ladrillo viene dada por su color original. La puntuación del jugador, y la posición de la pelota, se muestran por la cónsola o símbolo de sistema mientras se juega.

Organización del código

El código se encuentra dividido en nueve archivos .c, con sus respectivos .h, y un archivo de encabezado (header file) independiente. El propósito de cada uno de los mismos se describe brevemente a continuación, dejando la explicación de las funciones que contienen para la sección correspondiente.

valores_globales

valores_globales.h es un archivo de encabezado independiente. Contiene las definiciones de varias constantes requeridas por diversas clases en el código, así como declaraciones de variables globales importantes y definiciones de las estructuras lista_niveles, estructura_nivel

basico

basico.c y basico.h contienen funciones para dibujar las primitivas básicas requeridas por el programa.

cargarArchivo

cargarArchivo.c y cargarArchivo.h contienen la función que carga el archivo en las estructuras definidas en valores_globales

error_services

Las rutinas de error_services.c y error_services.h proporcionan métodos que permiten generar, almacenar, y reportar mensajes de error. Son usadas en caso de un error matemático, de memoria, o de comunicación con el sistema de archivos.

pelota

pelota.c y pelota.h definen las funciones de interacción de la pelota que detectan y simulan los efectos de las fuerzas físicas sobre la misma.

plano

plano.h define la estructura “plano” utilizada por pelota.c en sus funciones. La estructura “plano” será explicada en más detalle más adelante

ladrillo

En ladrillo.c y ladrillo.h se almacenan las funciones para manipular los ladrillos en pantalla.

ladrilloGL

ladrilloGL.c y ladrilloGL.h permiten dibujar los ladrillos en pantalla.

timer

El código en timer.c y timer.h está diseñado para computar intervalos de tiempo independientemente del sistema operativo. Ya que la función del estándar de C para obtener tiempos menores a un segundo (clock) no se implementa uniformemente en todos los sistemas

operativos, este código utiliza compilación condicional para asegurarse de que en Linux se use getTimeOfDay, en Windows se use GetClockTicks, y en cualquier otro caso, time. Esto permite que se obtenga el tiempo transcurrido con la mayor precisión posible, y es usado para el tiempo de enfriamiento.

enfriamiento

Las funciones de enfriamiento.c y enfriamiento.h implementan lo necesario para que, al cumplirse las condiciones de enfriamiento, los ladrillos sean movidos hacia abajo. Utiliza timer.h para verificar el tiempo transcurrido, y lo compara con lo que previamente habían guardado los métodos de cargarArchivo en las variables de valores_globales.

main

El programa principal carga el archivo usando las funciones de cargarArchivo.h, fija las funciones que dibujan la escena, y establece cuándo llamar a las funciones de pelota.h, ladrillo.h, y enfriamiento.h

Estructuras definidas

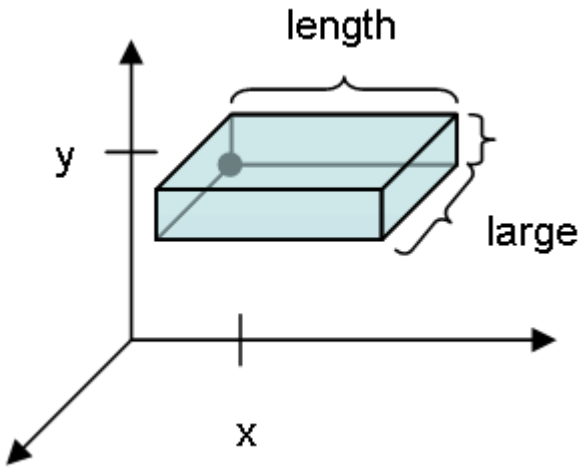
struct lista_niveles

Esta estructura de valores_globales.h define una lista de niveles. Se compone de un apuntador a una lista de apuntadores, y un entero que indica la longitud de esta lista, sirviendo como indicador de el número de niveles en el juego. Cada nivel apuntado es almacenado en la forma de un struct estructura_nivel.

struct estructura_nivel

Cada estructura_nivel se compone de una lista de apuntadores a caracteres, operada como una matriz n×m. El tamaño de la matriz viene dado por las constantes FILAS_LADRILLOS y ANCHO_MAXIMO, definidas, al igual que este struct, en valores_globales.h. A pesar de los ladrillos se encuentran en el archivo en un formato muy similar al usado internamente por el método dibujarLadrillos, se optó por esta codificación debido a que la cantidad de memoria que debe ser solicitada para cada nivel es constante, consistiendo de 1 byte por cada espacio del tablero, y no se sabe por cuánto tiempo deben ocupar memoria.

struct parallelepiped



La estructura parallelepiped, es una estructura que utilizamos para poder representar a los bloques que aparecen en el tablero del juego, es decir, es un simple paralelepipedo. Como paralelepipedo consta de las dimensiones de alto, largo, y ancho, estos valores son representados por high, large y length respectivamente. Ademas de los atributos de las dimensiones, se encuentran los atributos de color y resistencia. Como se especifico en el proyecto los ladrillos deben tener cierta resistencia asociada a un color, es por ello que se tiene el atributo resistencia y color.

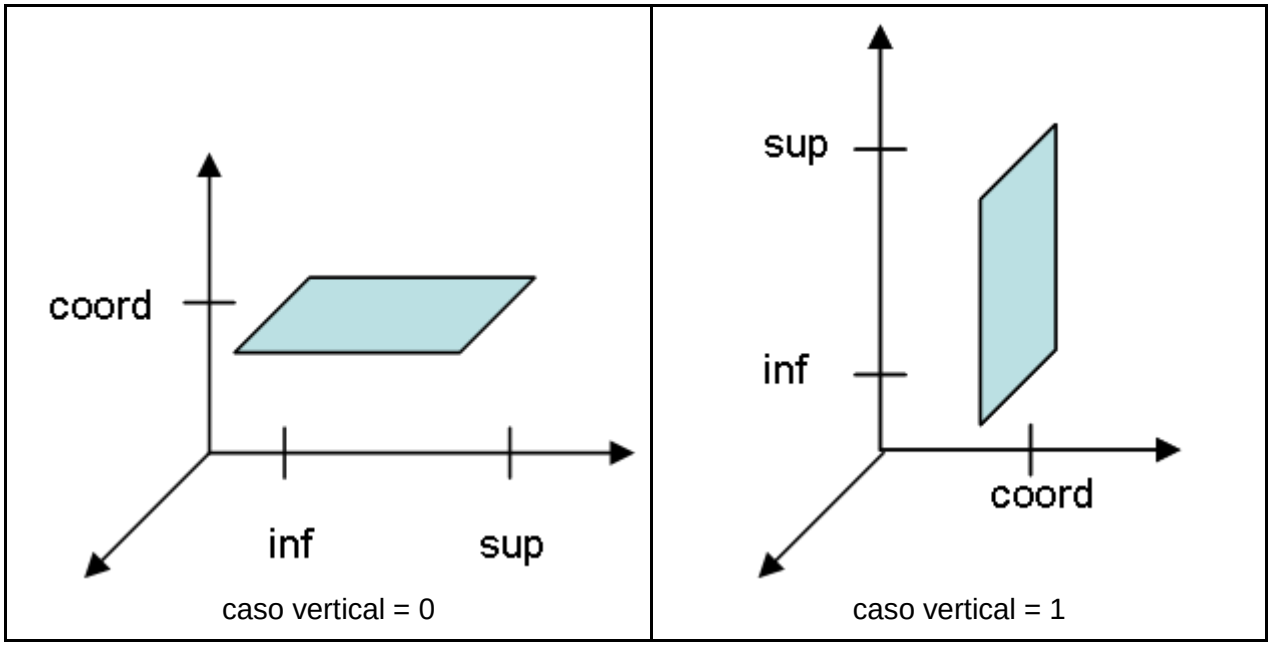
struct sphere

La estructura sphere,definida en la zona global se refiere a la pelota en el juego. Esta estructura representa las coordenadas de la esfera ademas de caracteristicas de la misma, como lo son su radio, velocidad en el eje x y velocidad en el eje y, y su aceleracion. Para crear la esfera se utilizo una instruccion de openGL llamada glutWireSphere, por lo que en esta estructura tambien se especifica el numero de subdivisiones de poligonos que se desea tenga la esfera, estos son los slices y stacks.

struct plano

Definido en plano.h, esta es la estructura usada para almacenar planos bidimensionales; principalmente las paredes del área de juego. La estructura almacena la información relevante para rebotar_pelota: la coordenada del plano en el eje al cual es perpendicular el plano, y sus fronteras inferiores y superiores en el eje al cual es paralelo. La dimensión restante (comúnmente llamado el ancho) no es relevante ya que todo movimiento se efectúa sobre el eje XY

Para poder representar los distintos tipos de plano con la misma estructura, se define el atributo “vertical”, el cual está definido para valores 0 y 1. Cuando “vertical” es 0, el plano es interpretado como paralelo al plano XZ. Cuando “vertical” es 1, el plano es interpretado como paralelo al plano YZ.



“main” crea un plano con vertical=2 para representar la pared posterior del tablero (paralela al plano XY). Sin embargo, este plano nunca es utilizado, por lo que el programa no tiene que lidiar con que su interpretación carezca de sentido.

Implementación de simulación física

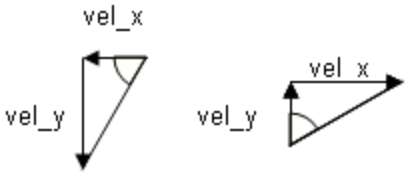
colision_plano

colision_plano en pelota.h verifica que la pelota esté en contacto con un plano, donde "contacto" puede variar de estar a punto de tocando el plano la siguiente vez que se refresque la pantalla, a estar incrustado en él. Verifica que la distancia entre el centro de la pelota y el plano (suponiendo que el segundo de estos es infinito) sea menor que el radio de la pelota y, si lo es, verifica que la pelota se encuentra dentro de los límites del plano. Si la distancia entre el centro de la pelota y el borde del plano es menor que el radio, retorna un valor distinto, ya que al impactar contra el mismo la simulación debe cambiar.

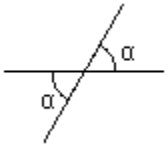
Ya que las posiciones son propensas a errores de redondeo, todas las comparaciones se hacen suponiendo un error entre cero y un épsilon definido al principio del archivo.

rebotar_pelota

Para permitir que la pelota pueda rebotar con cualquier superficie, sin importar su rotación en el eje Z, se definió rebotar_pelota en pelota.c en función de una sphere y un ángulo. Este método cambia la velocidad en X y en Y de la pelota recibida. Para calcular los nuevos valores de vel_x y vel_y, la función utiliza unas fórmulas que derivamos a mano usando el siguiente razonamiento:

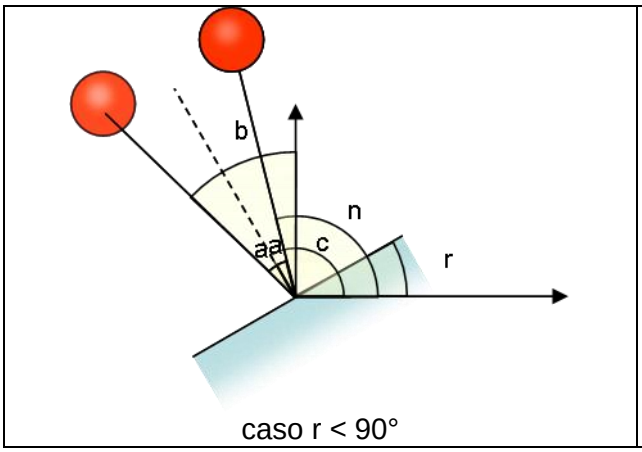


Imaginamos un triángulo formado por los vectores vel_x × $\hat{i}$ y vel_y × $\hat{j}$. Los valores a buscar (vel_x' y vel_y') crean un triangulo similar; sin embargo, para evitar que la energía de la pelota cambie – causando que parezca acelerar o desacelerar – se establece que el módulo del vector $\langle \text{vel\_x}', \text{vel\_y}' \rangle$ debe ser igual al del vector $\langle \text{vel\_x}, \text{vel\_y} \rangle$. Este módulo viene dado por la hipotenusa del triángulo.

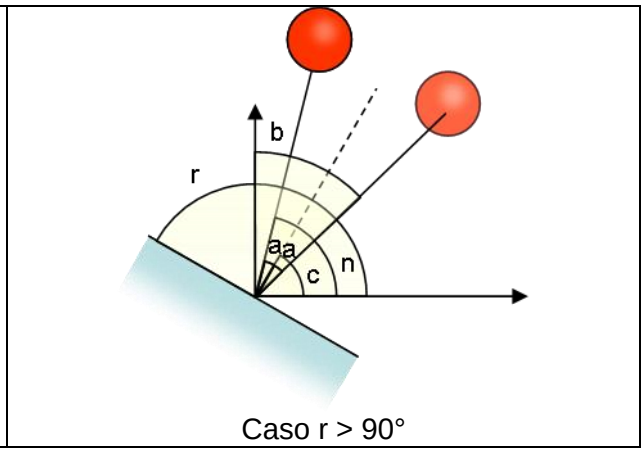


El ángulo de la trayectoria actual (n) con respecto a X es el de vel_x con la hipotenusa, debido a la regla mostrada anteriormente. Para hallar los ángulos del nuevo triángulo, es necesario encontrar el ángulo (b) de la trayectoria con respecto al eje Y.

Para calcular este ángulo, separamos en dos caso: cuando 0<r<90° y cuando 90°<r<180° (los casos r=90 y r=180 se cubren simplemente multiplicando la componente correspondiente por ¯1).



caso r < 90°



Caso r > 90°

Para el primero de estos, sabemos que c–a=n y que c+a–r = 90°.
Despejando, encontramos c = 45°+r-n y que b = 90°–(c+n)

En el segundo caso, vemos que n–2a+b = 90° y que r–90°+a = b
Despejando, tenemos que b =90° - (n+2r). Nótese que el resultado no depende del ángulo de incidencia c.

detectarColisionPelotaLadrillos

Esta función de ladrillo.c crea, para cada ladrillo, cuatro planos que son congruentes con los cuatro lados del ladrillo relevantes para verificación de colisiones, y revisa si la pelota ha colisionado con alguno de ellos llamando a colision_plano

colision_plano_final

Se define un plano imaginario (es decir, un plano que *no* es dibujado) en pantalla representando el límite inferior del tablero. Si la pelota colisiona con el mismo, se considera que el jugador ha perdido una vida.

mover_pelota

Este método sencillo de pelota.c aplica la función básica de física $x_t=x_{t-1}+v_x$ para actualizar la posición de la pelota en el tiempo.

Interacción con GLUT

drawParallelepiped

Definida en basico.c, este método crea seis planos, uno por cada cara del paralelepípedo dado.

drawSphere

drawSphere en basico.c parsea los atributos de un struct sphere y se los pasa a glutSolidSphere, dibujando la esfera especificada en su lugar

dibujarLadrillos

ladrilloGL.c consiste principalmente de esta función, la cual recorre la lista de ladrillos (almacenados como struct parallelepiped) y dibuja cada uno de su color apropiado llamando a drawParallelepiped

reproject

reproject es la función llamada por glut al ocurrir un evento “reshape”. Establece los límites del área de proyección.

drawScene

El corazón del programa, drawScene es llamado cada vez que se requiere un evento “display”. Además de dibujar los elementos en pantalla, verifica si la pelota colisionó contra la paleta, contra un ladrillo, o contra alguna de las paredes, en cuyo caso la hace rebotar de manera acorde.

Debido a que el incremento de velocidad tras un número de impactos con el tope del área incrementa la velocidad de forma impredecible, se establecen dos medidas de control para asegurase de que el juego siga siendo jugable: la primera es un límite de velocidad para la pelota, después del cuál es imposible para el ojo humano seguirla. La segunda es la verificación de si la pelota saltó la talanquera. Ésta última es necesaria ya que colision_pelota supone que la velocidad de la pelota es constante para poder saber si la pelota está a punto de colisionar con un objeto; al ocurrir una aceleración impredecible, esta suposición se hace falsa, invalidando los cálculos de la misma. No se considera necesaria una verificación secundaria de este tipo para el impacto con un ladrillo ya que, si la pelota llegase a traspasar una de las paredes de un ladrillo, se detectaría su colisión con la siguiente pared, permitiendo así computar el valor de dicho impacto.

Implementación de características fundamentales de *Breakout*

startLevel y cable

Ya que al inicio de un nivel la pelota se encuentra en reposo sobre la paleta, se introduce el concepto de “cable”, una variable precableada en el programa que indica si la pelota debe moverse con la paleta, o si debe moverse libremente en donde puede colisionar con otros objetos.

startLevel también convierte la matriz generada por cargar_archivo en la lista de struct parallelepiped requerida por las funciones de ladrillo.c.

lanzar_pelota

Esta función de pelota.c recibe un ángulo y una magnitud e inicializa las componentes vel_x y vel_y del vector velocidad, requeridas por las funciones de simulación de física. El ángulo es escogido al azar del conjunto {-5°, -4°, -3°, -2°, -1°, 0°, 1°, 2°, 3°, 4°, 5°} por startPlaying en main.c

checkLevelChange

Es necesario saber cuándo han sido destruidos todos los ladrillos, ya que en ese momento, debe cargarse el siguiente nivel. La función encargada de esta detección es checkLevelChange en main.c. También se encarga de indicarle al jugador que ha ganado cuando destruye el último ladrillo del último nivel

condiciones_finales

condiciones_finales revisa si se han dado las condiciones para cerrar el programa (ya sea por victoria o por Game Over). Devuelve 0 ó 1 y es responsabilidad de la función llamadora terminar el programa.

Implementación de especificaciones adicionales

revisarEnfriamiento

Para cumplir con la especificación que requiere que los ladrillos se muevan hacia abajo al haber transcurrido un tiempo, se define revisarEnfriamiento en enfriamiento.c el cual utiliza las funciones de timer.h para comprobar si ya ha transcurrido suficiente tiempo desde la última llamada a startLevel

Para mantener el tiempo de salto lo más uniforme posible, también utiliza las funciones de timer.h para medir el tiempo desde la última llamada a sí misma.

ladrilloEnLineaFinal

Siguiendo la especificación de que el juego debe terminar cuando los ladrillos lleguen a la línea final, se declara ladrilloEnLineaFinal en main.c. Éste utiliza el plano imaginario definido para drawScene y verifica si alguno de los ladrillos lo ha pasado.

stepDown

stepDown en enfriamiento.c hace el salto efectivo para todos los ladrillos en la escena

Conclusiones

BreakOut resulta un juego muy completo, en el sentido de que tiene varios grados de dificultad y reglas que lo hacen interesante y entretenido. Este juego contiene características que lo hacen atractivo en la programación. Por ejemplo, la detección de colisiones, el trato que se le debe dar a los ángulos, etc.
Además, requiere tener graficas que representen físicamente estas abstracciones. Y se necesita conocimiento de las leyes de la fisica para poder tener un juego realista y agradable visualmente.