Introduction to SDL

Vincent Chau, Geoff Leach

RMIT University

What is SDL?

- SDL – Simple Directmedia Layer
- `http://www.libsdl.org/`
- " Simple DirectMedia Layer is a cross-platform development library designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D."
- Used by Valve and in many Humble Bundle games
- Good subsitute for GLUT/FreeGLUT.
- Provides greater control and flexibility than GLUT.

Review of GLUT

- Simple interface/framework for writing simple OpenGL applications. Initialises OpenGL subsystem on our behalf and handles all interactions with the windowing system.
- Works on a system of programmer supplied callback functions, which are called by GLUT when the appropriate event occurs. eg. keyboard callback function is called when user presses a key.
- GLUT shields the programmer from the details of setting up OpenGL and the processing of events and so is a good platform for beginners. However in doing so it is also more restrictive with respect to applications such as games design.

Overview of SDL

- More complicated to use than GLUT, because it is lower level.
- Lower level in the sense that the programmer is expected to do more of the setup of OpenGL resources, main event loop, etc.
- In GLUT, programmer just supplies user input callbacks, simple initialisation code and a display function, and GLUT performs all the event checking and runs the main program loop. In SDL this is all handled by the programmer.
- Provides an API through which the programmer can perform these tasks (video, window manager, events, joysticks, audio, threads and timers).
- Gives the programmer more control over the way this functionality is handled. Means more work, but for a good cause!

What SDL Doesn't Do

- Doesn't have nice functions for drawing various geometric objects. Either do it yourself, use meshes, or use the functions from GLUT. Also consider using GLU quadrics.
- Doesn't draw menus or buttons or any nice UI stuff. You have to do that yourself.
- Doesn't do 3D sound or nice sound mixing. See OpenAL or FMOD for that functionality.

What You Need

- Many Linux distributions (Debian, Fedora, Ubuntu) come packaged with SDL. Need to make sure the development packages for SDL are installed.
- Cross platform: Windows, Mac OS X, Android
- Consists (mostly) of libSDL library and header files
- Documentation may be found on the SDL website `http://www.libsdl.org/intro/toc.html`
- SDL2 recently released (late 2013). Being used by Valve for its Linux games.
- We will use SDL2, although not that much difference for (relatively) simple usage.

GLUT Initialisation

- Regardless of whether GLUT or SDL is used, the steps of initialising the graphics device and the application window need to be performed.
- Robot arm (from the GLUT demos):

```
int main(int argc, char* argv[])
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  ...
```

SDL Initialisation

- `#include <SDL2/SDL.h>`
- SDL initialisation function needs to be called.
  `int SDL_Init(Uint32 flags);`
- Flags is used to identify the sub-systems that the programmer wishes to use: `SDL_INIT_TIMER`, `SDL_INIT_AUDIO`, `SDL_INIT_VIDEO`, etc. Also `SDL_INIT_EVERYTHING` and `SDL_INIT_NOPARACHUTE`.
- Do not have to initialise everything at once. Perhaps you need to do some other things before you initialise other sub-systems?
  `int SDL_InitSubSystem(Uint32 flags);`
  Allows you to initialize sub-system of your choice.

SDL OpenGL Initialisation

- Three SDL functions are used together to mimic the behaviour of the other 4 GLUT functions:

```
int SDL_GL_SetAttribute(SDL_GLattr attr, int value);
SDL_Window* SDL_CreateWindow(const char* title,
int x, int y, int w, int h, Uint32 flags);
SDL_GLContext SDL_GL_CreateContext(SDL_Window* window);
```

- `SDL_GL_SetAttribute()` is used to set various OpenGL resource attributes, including whether to use double buffering, and various framebuffer options.
- Once the desired attributes are set, `SDL_CreateWindow()` and `SDL_GL_CreateContext()` are used to create an application window and an OpenGL context. OpenGL calls cannot be made until this has been done.

Initialisation Example

```
SDL_Window *window;
SDL_GLContext glcontext;
SDL_Init(SDL_INIT_VIDEO);
SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
SDL_GL_SetAttribute(SDL_GL_DEPTH_SIZE, 16);
SDL_GL_SetAttribute(SDL_GL_RED_SIZE, 8);
SDL_GL_SetAttribute(SDL_GL_GREEN_SIZE, 8);
SDL_GL_SetAttribute(SDL_GL_BLUE_SIZE, 8);
SDL_GL_SetAttribute(SDL_GL_ALPHA_SIZE, 8);
window = SDL_CreateWindow("Robot Arm Example using SDL2",
  SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED, width, height,
  SDL_WINDOW_OPENGL | SDL_WINDOW_RESIZABLE));
if (window == NULL) {
  fprintf(stderr, "main: SDL_CreateWindow failed: %s\n", SDL_GetError());
  SDL_Quit();
  exit(1);
}
glcontext = SDL_GL_CreateContext(window);
```

Event Handling in GLUT

- GLUT has an internal loop that does the event processing for us. The programmer is never directly exposed to any events. The programmer simply supplies callbacks, which GLUT then executes when the appropriate event is received.
- `glutKeyboardFunc()` lets the programmer supply a function that is called when a regular key on the keyboard is pressed. `glutSpecialFunc()` is used for special keys such as the function keys and arrow keys.
- What about modifier keys (SHIFT, ALT, CTRL)? GLUT has a routine to check the current modifier state, however that is only updated when one of the normal or special keys is pressed in conjunction.

Event Handling in GLUT (Cont'd)

- Window manager events, such as the resizing of the window are automatically handled by GLUT, with the aid of an optional programmer supplied `glutReshapeFunc()`.
- Redrawing of the screen is automated with `glutDisplayFunc()`.
- eg.

```
int main(int argc, char* argv[])
{
  ...
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  ...
}
```

Event Handling in SDL

- SDL does not do any of the event processing for you. Instead it receives all the events from the window system and places them in a queue which you can access.
- The easiest way to do this is using
  `int SDL_PollEvent(SDL_Event *event);`
  which will return 1 if it found an event in the queue, and 0 otherwise. eg.

```
SDL_Event   e;
if (SDL_PollEvent(&e))
  printf("We got one!\n");
else
  printf("Move along now... nothing to see here.\n");
```

What's the Occasion?

- The `SDL_Event` type is a union of various structs, each of which is used for a different type of event.
- Determine the event type by checking `SDL_Event.type` and take appropriate action.

```
SDL_Event e;
if (SDL_PollEvent(&e))
{
  switch (e.type)
  {
    case SDL_QUIT: exit(EXIT_SUCCESS);
    case SDL_KEYDOWN: keyDown(&e.key); break;
    case SDL_WINDOWEVENT:
    ...
  }
}
```

Where the Party At?

- For events like SDL_KEYDOWN, examine the corresponding event struct for more detailed information. eg.

```
void keyDown(SDL_KeyboardEvent* e)
{
   switch (e->keysym.sym)
   {
      case SDLK_a: /* 'A' key */
         if (e->keysym.mod & KMOD_SHIFT)
         {  /* one of the shift keys was pressed */
            if (e->keysym.mod & KMOD_LSHIFT) printf("lshift\n");
            if (e->keysym.mod & KMOD_RSHIFT) printf("rshift\n");
         }
         break;
      case SDLK_LSHIFT: break;
      case SDLK_F1:     break;
      ...
   }
}
```

Why a Main Loop?

- So far we have looked at retrieving a single event and how to process it.
- A game we will have many events though.
- Also need to process events each frame.
- Speaking of frames, when do we draw them? What do we do with the display function?
- We need a loop that will iterate for the entire duration of the game that handles all of the above.
- GLUT has the `glutMainLoop()` function. For SDL we will write our own.

I Got Event, I Got Event, I Got Event, YAYE!

- Earlier we retrieved a single event. We can modify that to retrieve all of the available events and handle them as necessary. ie. An event dispatcher:

```
while (SDL_PollEvent(&e))
{
    switch (e.type)
    {
        case SDL_QUIT:    exit(EXIT_SUCCESS);
        case SDL_KEYDOWN: keyDown(&e.key);break;
        ...
    }
}
```

- If we then append the code to draw the scene, and wrap it up in one big loop, we end up with the following slide.

Main Loop Example

```
void mainLoop()
{
   SDL_Event   e;
   while (1)
   {
      while (SDL_PollEvent(&e))
      {
         switch (e.type)
         {
            ...
         }
      }
      display();
   }
}
```

Improvements

- The above example could be likened to a GLUT program with: glutIdleFunc(display);
  Do you see why?
- Rather than calling display() directly, GLUT provides `glutPostRedisplay()`, which is used to queue the `display()` function. Ensures that at most one redraw is performed per iteration of the outer loop, and possibly none.
- We can replicate this behaviour with the aid of a global variable and a function.

```
static int  wantRedisplay = 0;
void postRedisplay() { wantRedisplay=1; }
```

```
void keyDown(SDL_KeyboardEvent* e)
{
   ...
   /* Do something that requires a redraw */
   postRedisplay();
   ...
}

void mainLoop()
{
   while(1)
   {
      eventDispatcher();
      if (wantRedisplay)
      {
         display();
         wantRedisplay = 0;
      }
      idle();
   }
}
```

Other GLUT to SDL Mappings

GLUT	SDL
`void glutSwapBuffers()`	`void SDL_GL_SwapWindow()`
`void glutWarpPointer (int x, int y)`	`void SDL_WarpMouseGlobal (Uint16 x, Uint16 y)`

Resizing and Unhiding the Window

- If you are not running fullscreen, inevitably the window will get resized, or hidden. There are events for these too.

```
SDL_Window*  window; /* Set during initialisation */

void eventDispatcher()
{
  SDL_Event e;
  ...
  switch (e.type)
  {
    ...
    case SDL_WINDOWEVENT:
      switch (e.window.event)
      {
        case SDL_WINDOWEVENT_EXPOSED: postRedisplay(); break;
        case SDL_WINDOWEVENT_RESIZED: resize(&e.window);
        ...
      }
  }
```

Resizing and Unhiding the Window (Cont'd)

- A resize event requires us to use `SDL_SetWindowSize()`, we can then perform our usual `reshape()`, and redraw.

```
void resize(SDL_WindowEvent* e)
{
  if (e->window.windowID == SDL_GetWindowID(window)) {
    SDL_SetWindowSize(window, e->window.data1, e->window.data2);
    reshape(e->window.data1, e->window.data2);
  }
   postRedisplay();
}
```