3 Multimedia Programming with C++ and Multimedia Frameworks

3.1 Multimedia Support by Languages and Frameworks
3.2 Introduction to C++
3.3 SFML: Low-Level Multimedia/Game Framework for C++
3.4 Cocos2d-x: High Level Game Engine for C++
Multimedia Support is Platform Dependent

- Example: 2D graphics on desktop/laptop computers
Multimedia Abstraction Layer

Multimedia Application

Multimedia Abstraction Layer

Examples: SDL (Pygame), SFML

Direct X
XLib
Quartz

Windows
Linux
OS X

Hardware

System-level graphics software

Operating system

Computer/mobile device
Graphics card
Core Functionality of a Multimedia Abstraction Layer

**Graphics**
- Surfaces
- Vector Drawing
- Bitmap Images

**Sound**
- Recording and playing
- Mixing
- Sound files and streams

**Moving Images**
- Video playback
- Moving 2D images (sprites)
- Translations, transformations

**Input/Output and Events**
- External devices
- Event queuing

**Network**
- Sockets
- Protocol support (FTP, HTTP)

**Time**
- Clock
- Timers
Additional Functionality for Advanced Multimedia Applications (Games)

**Object Structure**
- Layers, scenes
- Scene graphs
- ...

**Control Structure**
- Activity scheduling
- Time containers
- ...

**Animation**
- Animated images
- Sprite sheet support
- ...

**Effects**
- Interpolators
- Complex transitions
- ...

**Physics Simulation**
- Solid-body physics
- Particle effects
- ...

**Artificial Intelligence**
- Game logic
- Adaptation
- ...
High-Level Multimedia Framework

Multimedia Application

Built-in Abstraction Layer

System Libraries

OS

Hardware

Advanced Functionality

High-level multimedia framework

Examples: Cocos2d-x, …
Terminology

• Low-Level Multimedia Framework
  = Multimedia Abstraction Layer
  – Platform independent media handling
• High-Level Multimedia Framework
  = Game Engine
  – Advanced multimedia features
  – Specific support for game functionalities
  – In most cases includes low-level multimedia framework
Program Control and Frameworks

**Main**:
- `<all preparation work>`
- while running do:
  - `<process input>`
  - `<update model>`
  - `<render model>`

**def init()**:
- `<specific preparations>`

**def update()**:
- `<specific updates>`

```python
schedule(this)
```

**Traditional program control**
- We write the main program
- Main control loop is part of our code
- We call functions of the framework

Typical for low-level frameworks

**Inversion of program control**
- Main program is in the framework
- Main control loop does not appear in our code
- The framework calls our functions
- "Hollywood Principle"

Typical for high-level frameworks
Implementation Language: Criteria

• Handling of multimedia objects
  – Built into the language?
  – Available in standard library? (e.g. Yes for Java 8)
• Portability to target platforms
  – Critical in case of multiple platforms, mobile platforms etc.
• Performance: Memory efficiency
• Performance: Fast execution
• Performance: Access to hardware accelerations
• Performance…
# Python, Java, C++

<table>
<thead>
<tr>
<th></th>
<th>Python</th>
<th>Java</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ease of use</strong></td>
<td>easy to learn and use</td>
<td>medium complexity</td>
<td>rather demanding</td>
</tr>
<tr>
<td><strong>Multimedia support</strong></td>
<td>not standard available through frameworks (Pygame)</td>
<td>Java 8: well-supported through built-in framework (JFX)</td>
<td>not standard available through frameworks (SDL, SFML, Cocos2d, …)</td>
</tr>
<tr>
<td><strong>Multi-platform</strong></td>
<td>basically given limitations regarding multimedia frameworks very limited for mobile devices (being improved by e.g. kivy.org)</td>
<td>basically OK single-source multi-platform limited e.g. for iOS</td>
<td>well supported through frameworks (e.g. Cocos2d-x)</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>limited</td>
<td>limited</td>
<td><strong>very good</strong> (memory, native, hardware acceleration)</td>
</tr>
</tbody>
</table>
3 Multimedia Programming with C++ and Multimedia Frameworks

3.1 Multimedia Support by Languages and Frameworks
3.2 Introduction to C++
3.3 SFML: Low-Level Multimedia/Game Framework for C++
3.4 Cocos2d-x: High Level Game Engine for C++

Literature:
B. Stroustrup: The C++ programming language, 4th ed., Addison-Wesley 2013
C++: History

- C: Dennis Ritchie, Bell Labs, 1969-1973, an improvement of B, which was a simplified version of BCPL
- Bjarne Stroustrup, Bell Labs, 1979: Extension of the C programming language for large-scale programming *(16 years before Java!)*
- ISO standard since 1998 (ISO/IEC 14882)
- Important late revisions and extensions, in particular C++11 (2011)
- C++17 in preparation
- Main features of the language:
  - Object-oriented programming with multiple inheritance
  - Flexible storage allocation models
  - Templates for generic programming
  - Lambda expressions
  - Exception handling
  - Powerful standard library
Differences Java – C Type Languages

• **Good news first:**
  Syntax is very similar!

• Native compilation (C++) vs. compilation to VM (Java)

• Not everything has to be a class in C / C++.

• Preprocessor instead of import
  
  ```
  #include <string>
  using namespace std;
  ```

• Much more flexible (and dangerous) concept of pointers (and references):
  Pointers may refer to any storage area, in particular to memory on the stack!

• Objects can be created in C++ on the stack (static memory), not only on the heap (dynamic memory).

• Allocated memory on the heap (new) does **not** have automatic garbage collection. Memory has to be freed explicitly by "delete".
Pointers and References in C++

- Address-of Operator (&)
  ```
  int myvar = 25;
  &myvar is the memory address of myvar (on the stack)
  ```

- Pointer type declaration:
  ```
  int* ptr = &myvar;
  Printing ptr shows a memory address (e.g. 0x7fff5fbff698)
  ```

- Dereference Operator (*)
  ```
  *ptr is the value at the address to which ptr currently points
  Printing *ptr gives the value of myvar
  ```

- References (rarely used):
  ```
  int& ref = myvar;
  ref is like an alias of myvar and cannot be re-assigned
  ```
Multiple Files, Header Files

• Using an additional cpp file:
  – Create additional header file:
    Counter.cpp and Counter.hpp

• Header file:
  – Contains declarations, is used to refer to the contents of the additional file
  – In Counter.cpp and wherever Counter.cpp is used (e.g. in Main.cpp):
    #include "Counter.hpp"

• Macros needed to avoid multiple declarations:
  #ifndef Counter_hpp
  #define Counter_hpp
  // Declarations go here
  #endif /* Counter_hpp */
Counter Class in C++, Header File

```cpp
#ifndef Counter_hpp
#define Counter_hpp

class Counter {
    int k;
public:
    Counter();
    void count();
    int getValue();
};

#endif /* Counter_hpp */
```

Note: Using one pair of files for a class is just a recommendation – not required by C++
Counter Class in C++, Body File

#include "Counter.hpp"

Counter::Counter() {
    k = 0;
}

void Counter::count() {
    Counter::k++;
}

int Counter::getValue() {
    return Counter::k;
}
Counter Class in C++, Main Program

```cpp
#include <iostream>
#include "Counter.hpp"

int main(int argc, const char * argv[]) {

    // New counter instance (on stack)
    Counter ctr;
    std::cout << "Value of ctr: " << ctr.getValue()
    << std::endl;

    ctr.count();
    std::cout << "Value of ctr: " << ctr.getValue()
    << std::endl;

    return 0;
}
```

main.cpp
C++: Scope Resolution and Namespaces

• Scope operator (::) of C++:
  – Various purposes

• First main purpose:
  – Implementing member functions of classes outside the actual class
  – Example Counter::count()
  – In this case, only the function prototype is declared within the class

• Second main purpose:
  – Qualifying identifiers as belonging to a certain namespace
  – Example std::cout which belongs to the std namespace

• Declaring a new namespace:
  namespace ns_name { declarations ;

• Using a given namespace for unqualified identifiers:
  using namespace ns_name;
3 Multimedia Programming with C++ and Multimedia Frameworks

3.1 Multimedia Support by Languages and Frameworks
3.2 Introduction to C++
3.3 SFML: Low-Level Multimedia/Game Library for C++
3.4 Cocos2d-x: High Level Game Engine for C++

Literature:
http://www.sfml-dev.org
M. G. Milchev: SFML Essentials, Packt Publishing 1015
SFML: Simple and Fast Multimedia Library

- Relatively modern multimedia abstraction layer for C++
- Initiator and head of development: Laurent Gomila (France)
- Latest version: 2.3.2 (September 2015)

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio module</td>
<td>Sounds, streaming (musics or custom sources), recording, spatialization</td>
</tr>
<tr>
<td>Graphics module</td>
<td>2D graphics module: sprites, text, shapes, ..</td>
</tr>
<tr>
<td>Network module</td>
<td>Socket-based communication, utilities and higher-level network protocols (HTTP, FTP)</td>
</tr>
<tr>
<td>System module</td>
<td>Base module of SFML, defining various utilities</td>
</tr>
<tr>
<td>Window module</td>
<td>Provides OpenGL-based windows, and abstractions for events and input handling</td>
</tr>
</tbody>
</table>
Slide Show with SFML (1)

```cpp
#include <SFML/Graphics.hpp>
#include "ResourcePath.hpp"
#include <array>

//Global dimensions
const int SCREEN_WIDTH = 712;
const int SCREEN_HEIGHT = 712;
const int OFFSET_HOR = 100;
const int OFFSET_VER = 100;

//Directory name for pictures
const std::string gPicsDir = "pics";

//Picture file names
const int NUM_PICS = 4;
std::array<std::string, NUM_PICS> gPicFiles =
    {"frog.jpg", "cows.jpg", "elephant.jpg", "tiger.jpg"};

const sf::Color bg = sf::Color(255, 228, 95, 255);  //background color
const sf::Time interval = sf::seconds(4.);
...
```
Slide Show with SFML (2)

...  
int main(int, char const**) {  
    // Create the main window  
    sf::RenderWindow window  
        (sf::VideoMode(SCREEN_WIDTH, SCREEN_HEIGHT), "SFML Slide Show");  

    //Load pictures  
    sf::Texture loadedPics[NUM_PICS];  
    for(int i = 0; i < NUM_PICS; i++) {  
        std::string picPath = resourcePath()  
            + gPicsDir + "/" + gPicFiles[i];  
        if (!loadedPics[i].loadFromFile(picPath)) {  
            return EXIT_FAILURE;  
        }  
    }  

    //Create a sprite to display  
    sf::Sprite sprite;  
    sprite.setScale(sf::Vector2f(2.f, 2.f));  
    sprite.setPosition(sf::Vector2f(OFFSET_HOR, OFFSET_VER));  

    ...
//Main loop
int slideIndex = 0; //Index of picture to be shown
bool updatePicture = true; //Do we need to change the picture?
sf::Clock clock; //Create and start timer
sf::Time timer;

while (window.isOpen()) {
    //Display next picture if necessary
    if (updatePicture) {
        //Clear screen
        window.clear(bg);
        //Set picture and draw the sprite
        sprite.setTexture(loadedPics[slideIndex]);
        window.draw(sprite);
        //Update the window
        window.display();
        updatePicture = false;
        // Set timer
        timer = clock.getElapsedTime();
    }
}

...
Slide Show with SFML (4)

// Process events
sf::Event event;
while (window.pollEvent(event)) {
    // Close window: exit
    if (event.type == sf::Event::Closed) {
        window.close();
    }
    // Arrow left pressed: previous picture
    if (event.type == sf::Event::KeyPressed
        && event.key.code == sf::Keyboard::Left) {
        slideIndex = (slideIndex+NUM_PICS-1) % NUM_PICS;
        //Strange C++ modulo
        updatePicture = true;
    }
    // Arrow right pressed: next picture
    if (event.type == sf::Event::KeyPressed
        && event.key.code == sf::Keyboard::Right) {
        slideIndex = (slideIndex+1) % NUM_PICS;
        updatePicture = true;
    }
}

...
Slide Show with SFML (5)

...  
  // Check time interval
  if (clock.getElapsedTime() > timer+interval) {
    slideIndex = (slideIndex+1) % NUM_PICS;
    updatePicture = true;
  }
}

    return EXIT_SUCCESS;
}