

---

# *Introduction to Panda3d, C++, and Python*

Kim Steenstrup Pedersen

`kimstp@itu.dk`

`www.itu.dk/courses/SSPG/F2005/`

The IT University of Copenhagen



# Programming languages for games

Favourite choices are: C / C++ and assembler language.

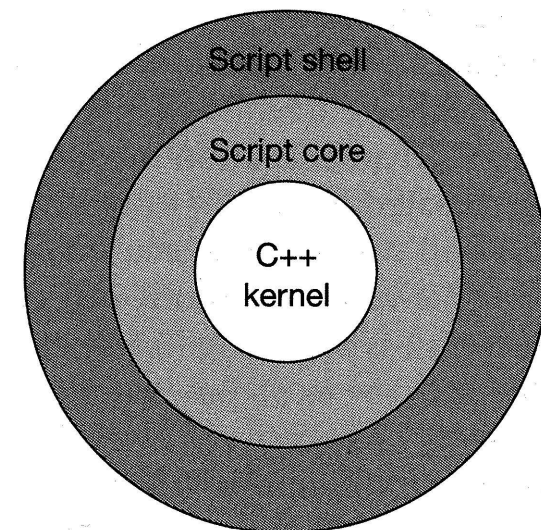
Maybe Java for mobile devices or web based games.

The choice of language should depend on what you are trying to achieve!

Different languages are in use on different platforms, so get used to learning new languages!

Scripting / interpreted high level languages are also important:

Good for rapid prototyping, and in some cases for production code.



Interpreted languages versus compiled languages.

# *Programming paradigms*

---

**Procedural:** Fortran, C, Pascal, (C++, Python)

**Object-Oriented:** C++, Python, Java, Modula, Smalltalk

**Functional:** LISP, ML

We will follow the object-oriented paradigm in this course and focus on C++ and Python.



# *Key elements of object-oriented programming*

---

- Data abstraction
- Encapsulation of data
- Inheritance
- Polymorphism



# *Game program design*

---

Key issues: Code reusability and rapid iterative development.

Achieved by using:

- Modular design
- Object-oriented design



# Game engines

---

In the spirit of code reuse, game engines / API's (Application Programming Interface) have become increasingly popular.

Definition of a game engine [J. Gold, 2004]:

*A series of modules and interfaces that allows a development team to focus on product game-play content rather than technical content.*

In this course we use the open source game engine called Panda3d.

Panda3d = Platform Agnostic Networked Display Architecture

Developed mainly by Disney VR Studio and Entertainment Technology Center at Carnegie-Mellon University. Disney use it among other things for the online game Toontown.

Examples of other game engines:

- Unreal
- Quake
- Torque
- Irrlicht (open source)



# Game engines: Panda3d

---

At the core Panda3d is a set of C++ libraries with functionality for:

- 2D and 3D visualisation
- Animation
- Sound
- Physics
- Networking
- I/O

All in a device independent manner to ensure painless cross platform development.

Supported platforms:

- Windows
- Linux
- Mac
- Sony playstation (not included in open source version)

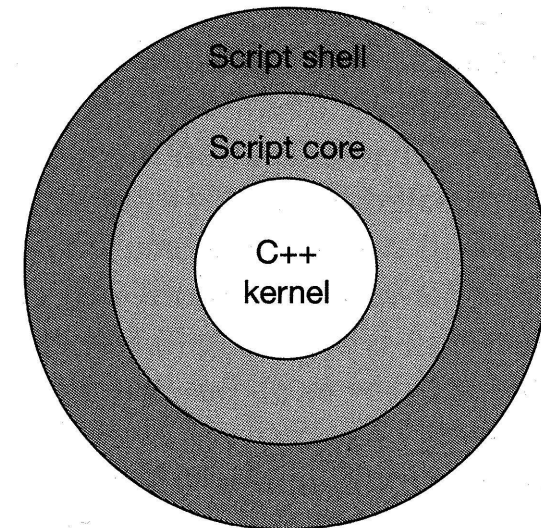
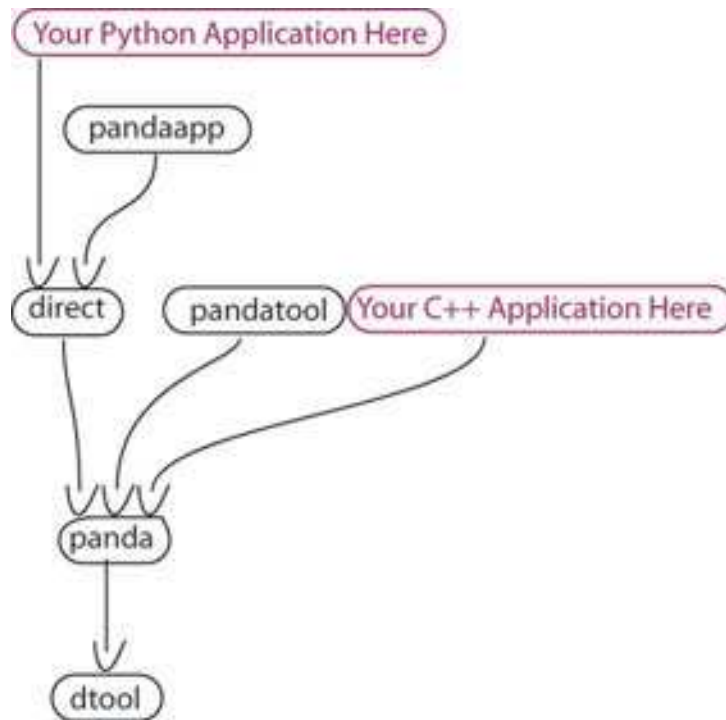
Graphics supported via API's:

- OpenGL
- Direct3D (DirectX)



## Game engines: Panda3d

On top of the C++ core is a Python interface, that allows for use of Panda functionality in Python scripts.



Show the Panda3d-Python demo.



## About C++

---

### Useful books:

- A.Koenig, B.E.Moo: **Accelerated C++**, 2001, Addison Wesley
- Stroustrup: **The C++ Programming Language, Special Edition**, 1997, Addison-Wesley

### Advice:

1. Don't panic! All will become clear in time:
2. You don't have to know every detail of C++ to write good programs
3. Focus on programming techniques, not language features

Stroustrup, p. 43



# Hello World!

---

```
// a small C++ program
#include <iostream>

int main()
{
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

## Things to note:

Comments, standard library, `#include`, `main`, curly braces, output operator `<<`, namespace, standard output stream, return statement, expression, side-effect, operand type, manipulator, scope and scope operator `::`.



## ***Building a C++ program***

---

**Building** = Compiling and linking your C++ program.

**Compiling:** Your C++ program is converted to one or more binary object files (.obj or .o file). Not runnable.

**Linking:** Your programs object files are combined with system and other types of libraries into a runnable program. Under windows this leads to a .exe file. Linking can also produce non-runnable new libraries, which are either static (.lib) or dynamic (.dll).

**Static library (.lib)** = linked with program at compile time.

**Dynamic library (.dll)** = linked with program at run time.

What is the benefit of this?



## Flow control statements in C++

---

A block defines compound statements and scope of local variables:  
`{ statements }`

`while (condition) statement`

`for (init-statement; condition; expression) statement`

`if (condition) statement`

`if (condition) statement else statement2`



# Defining variables and scope of variables

---

Example of defining variables:

```
{
    int number;
    std::string name;
    {
        double a = 1.5;
        std::string greeting = "Hello World!";
        const string spaces(greeting.size(), ' ');
    }
    a = 5.0; // Error!
}
```

Global variables:

```
int a; // Dangerous!
```

```
int main() {
    a = 2;
    return 0;
}
```

Things to note:

Variable, object vs built-in type, definition, local variable (scope of a variable), global variable, interface, (default) initialisation, `const`, constructor, member function, character literal, `char`.



## ***Build-in types are not objects***

---

Examples of build-in types in C++ are:

```
char           : -128 - 127 (8 bits)
unsigned char  :  0 - 255 (8 bits)
int, unsigned int : 32 bits
long long     : 64 bits
float         : 32 bits
double        : 64 bits
```

```
int a; // Value undefined
```

Things to note:

Not objects, no default initialisation.



## Frame2

---

```
[kimstp@haldir chapter02] ./frame  
Please enter your first name: Kim
```

```
*****  
*           *  
* Hello, Kim! *  
*           *  
*****
```



## Frame2

---

```
#include <iostream>
#include <string>

// say what standard-library names we use
using std::cin;           using std::endl;
using std::cout;         using std::string;

int main()
{
    // ask for the person's name
    cout << "Please enter your first name: ";

    // read the name
    string name;
    cin >> name;

    // build the message that we intend to write
    const string greeting = "Hello, " + name + "!";

    // the number of blanks surrounding the greeting
    const int pad = 1;
```



## Frame2 (cont.)

---

```
// the number of rows and columns to write
const int rows = pad * 2 + 3;
const string::size_type cols = greeting.size() + pad * 2 + 2;

// write a blank line to separate the output from the input
cout << endl;

// write 'rows' rows of output
// invariant: we have written 'r' rows so far
for (int r = 0; r != rows; ++r) {

    string::size_type c = 0;

    // invariant: we have written 'c' characters so far
    // in the current row
    while (c != cols) {

        // is it time to write the greeting?
        if (r == pad + 1 && c == pad + 1) {
            cout << greeting;
            c += greeting.size();
        } else {
```



## Frame2 (cont.)

---

```
        // are we on the border?
        if (r == 0 || r == rows - 1 ||
            c == 0 || c == cols - 1)
            cout << "*";
        else
            cout << " ";
            ++c;
        }
    }

    cout << endl;
}

return 0;
}
```

Things to note:

using declaration, cin, string concatenation, operator overload, while statement, block, condition, boolean operators, if statement, precedence, compound-assignment, for statement.



# STL = *Standard Template Library*

---

STL provides you with a lot of useful types, container classes, and algorithms. All this is in the namespace called `std`.

Examples of types and containers:

```
#include <iostream> ... std::cout
#include <string>    ... std::string
#include <vector>   ... std::vector
#include <list>     ... std::list
```

Examples of algorithms working on containers:

```
#include <algorithms>
...
sort(...);
find(...);
```

Useful, but watch out for varying performance between implementations of STL. More about this later in the course!



## Functions and overloading

---

```
// Loads a model from an input stream
int loadModel(std::istream& in, std::vector<Vertex>& model) {
    double x, y, z;
    while (in >> x >> y >> z) {
        Vertex v(x,y,z);
        model.push_back(v);
        std::cout << "Push back" << std::endl;
    }
    if (model.size() == 0) {
        std::cout << "Could not read model" << std::endl;
        return 1;
    } else
        return 0;
}

// Loads a model from file named filename
int loadModel(std::string filename, std::vector<Vertex>& model)
{
    ...
}

// Loads the default model
void loadModel(std::vector<Vertex>& model) { ... }
```



## Functions and overloading

---

```
int main() {
    std::vector<Vertex> model;
    if (loadModel(std::cin, model))
        return 1; // Error occurred

    for (int i=0; i != model.size(); i++) {
        std::cout << model[i].x << std::endl;
        std::cout << model[i].y << std::endl;
        std::cout << model[i].z << std::endl;
    }
    return 0;
}
```

Things to note:

Function definition, overloading, call by reference &, call by value, container template class `std::vector<T>`.



## Classes and methods

---

In header file `modelclass.h`:

```
class ModelClass {
public:
    ModelClass() {}; // Default constructor

    // Loads the default model
    void loadModel();

    // Loads a model from an input stream
    int loadModel(std::istream& in);

    // Loads a model from file named filename
    int loadModel(const std::string& filename);

private:
    std::vector<Vertex> vertices;
};
```

Things to note:

Class definition, `public:`, `private:`, `const` reference, splitting files (`.h`, `.cpp`, `.C`, `.cxx`), **avoid using `using` in header files!**



## Classes and methods

---

Implementation goes into `modelclass.cpp` file:

```
#include "modelclass.h"
```

```
using std::istream; using std::cout; using std::endl;
```

```
int ModelClass::loadModel(istream& in) {  
    double x, y, z;  
    while (in >> x >> y >> z) {  
        Vertex v(x,y,z);  
        vertices.push_back(v);  
    }  
    if (vertices.size() == 0) {  
        cout << "Could not read model" << endl;  
        return 1;  
    } else  
        return 0;  
}
```

```
void ModelClass::loadModel() { ... }
```

```
int ModelClass::loadModel(const std::string& filename) { ... }
```



## Using our class

---

```
#include "modelclass.h"

int main() {
    ModelClass model;

    if (model.loadModel(std::cin))
        return 1; // Error occurred

    return 0;
}
```

Things to note:

This program should be linked with the `modelclass.obj` file.



## Using Panda3d in C++

---

Missing C++ documentation for Panda3d! Not a problem because we will mix Python with C++ code that is independent of Panda. We will reverse engineer the parts of the C++ interface for Panda3d that are necessary.

But here is a code example:



## Using Panda3d in C++

---

```
#include <pandaFramework.h>

int main(int argc, char *argv[]) {
    PandaFramework framework;
    framework.open_framework(argc, argv);
    framework.set_window_title("Panda Helloworld");
    WindowFramework *window = framework.open_window();
    window->enable_keyboard();
    window->setup_trackball();
    framework.get_models().instance_to(window->get_render());
    if (argc < 2) {
        window->load_default_model(framework.get_models());
    } else {
        window->load_models(framework.get_models(), argc, argv);
    }
    window->loop_animations();
    framework.main_loop();
    framework.report_frame_rate(nout);
    return 0;
}
```



# Introduction to Python: Helloworld

---

Python is a script/interpreted language with an interactive interface.

In the file `helloworld.py`:

```
# This is a comment  
print "Hello, World!\n"
```

Lets try to run the program as a script and in interactive mode.

Start python by issuing the command `python` in a command prompt.

Get help about Python objects by typing `help(<name>)`.

Example: `help(help)`.



## *Introduction to Python: Types and syntax*

---

Variables does not have to be declared (defined with a type), you just have to use them!

```
a = 1
a = a + 1
print a
str = "This is a string"
print str
```

Python supports the same numerical types as in C++ as well as complex numbers.

Strings are defined by enclosing the text in " or ' .

Blocks of statements are defined by indentation of the lines of codes forming the block.



# Python functions and flow control

---

```
def foo(n): # Defining the function
    i = 0
    while i < n:
        print "Foo!"
        i = i + 1
    return i
```

```
a=foo(10) # Using the function
```

```
if a == 10:
    print a
```

```
b = [1, 2, 3]
for x in b:
    print x
```

Things to note:

def, while, return, if, for.



## Using Panda3d in Python

---

A simple example:

```
import direct.directbase.DirectStart

#Load the first environment model
environ = loader.loadModel("models/Environment")
environ.reparentTo(render)
environ.setScale(0.25,0.25,0.25)
environ.setPos(-8,42,0)

#Run the tutorial
run()
```

Run program! Move around in the scene.



## Overview of today

---

- Game development is not unique! We can apply good practises from other areas of computer science.
- We take an object-oriented approach to game development in the spirit of code reusability and rapid development.
- We use the Panda3d game engine/API.
- Panda3d is at the core a C++ library with a python interface wrapped around it.
- We will write extensions to Panda in C++ which can be used within Python.
- We will use Python as the language for gluing the components of our game together.



## Assignment for next lecture

---

No exercises today due to the intro week!

Your assignment for next time will be to read the material stated on the home page as well as solving the following small exercises:

1. Write and build a C++ helloworld program in MS Visual Studio .NET (2003). Create an empty C++ project to build a Win32 console application. Then add a main.cpp file to this project. Write the code in main.cpp.
2. Install the Panda library in your ITU home directory (see the course home page for more information).
3. Read and implement the Panda-Python tutorial found in the Panda documentation. Experiment with the code.
4. Experiment with Python.
5. Write a C++ class that implements a simple queue. That is, a first in first out (FIFO) list. (Hint: You can use `std::vector`. The best solution will require `std::list` and iterators. More about this next week.)



## Reading material

---

Reading material for this lecture:

- JG: Ch. 1-3
- Panda tutorial (first part of Panda3d documentation)
- KM: Ch. 0-6, 9
- Python tutorial

Reading material for the next lecture:

- JG: Ch. 3
- Parts of Panda documentation (to be announced)
- KM: Ch. 5-6, 9-11, 13
- Python tutorial
- Python extension documentation

The list of reading material for each lecture will be available on the course home page.



The slides will be made available on the course home page after the lectures.