Tutorial n°5 - Working with pointers and arrays of pointers

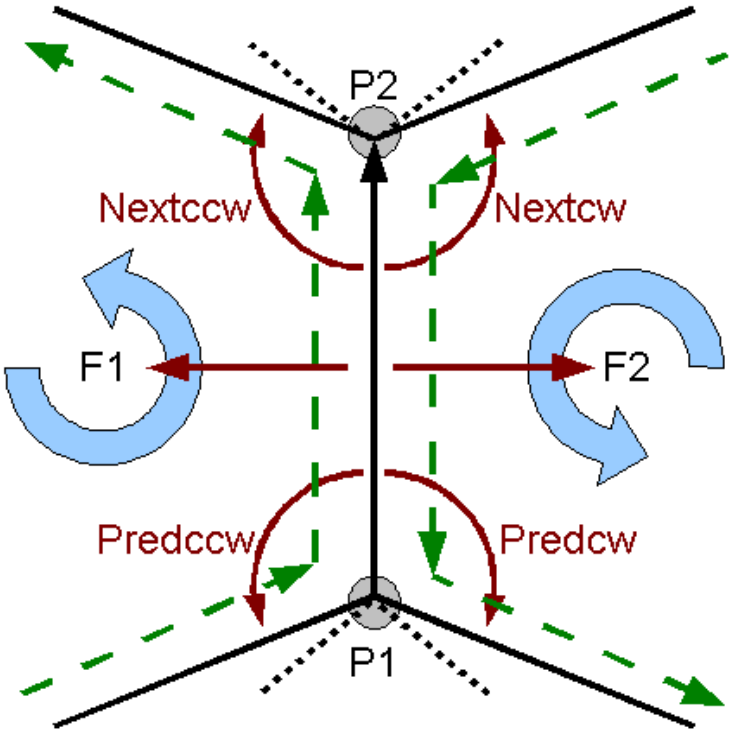
Application to $3D$ mesh representation

We have seen how to create arrays, pointers, double chained lists, and a simple example of application to represent $2D$ polygons. The focus of this labs is to introduce the concept of $3D$ Mesh and a possible solution to implement such a concept in $C++$. Compared to the previous labs, most of the code has already been implemented using CodeBlocks, so please use this IDE for this labs.

Objective: representation of a cube by Winged-Edge

Introduction

Perhaps the oldest data structure for a boundary representation (B-rep) is Baumgart's winged-edge data structure (1972). It is quite different from that of a wireframe model, because the winged-edge data structure uses edges to keep track almost everything. **In what follows, we shall assume there is no holes in each face.**



The above figure shows an edge $P1P2$. This edge has two incident vertices $P1$(start) and $P2$(end), and two incident faces $F1$ and $F2$. A face is a polygon surrounded by edges. Note that the ordering is counterclockwise viewed from outside of the solid. If the direction of the edge is from $P1$ to $P2$, $F1$ is on the left side of the edge $P1P2$, and $F2$ is on its right side.

To capture the ordering of edges correctly, we need four more pieces of information. Since edge $P1P2$ is traversed once when traversing face $F1$ and traversed a second time when traversing face $F2$, it is used twice in different (and opposite) directions. By convention, when traversing the edges of face $F1$ **counterclockwise**, the predecessor and successor of edge $P1P2$ are edges *Predccw* and *Nextccw*, respectively.

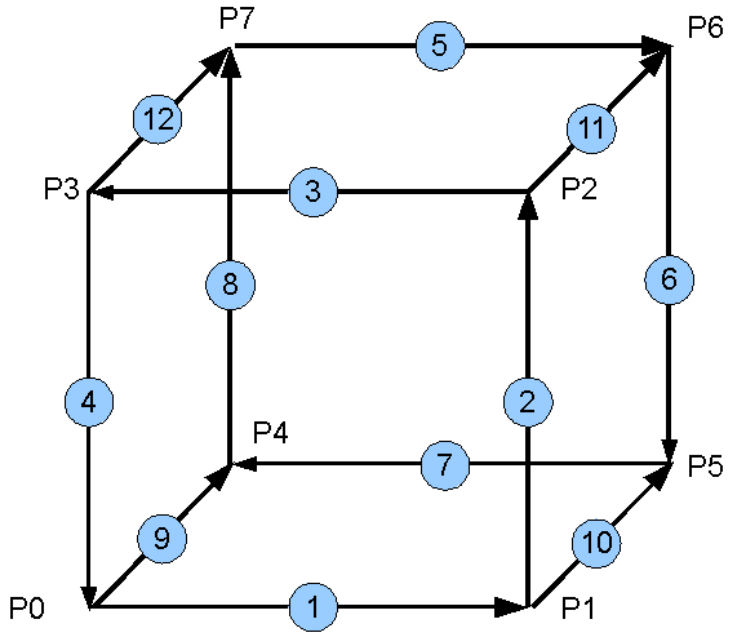
The successor of P1P2 in F2, i.e. the face that is traversed by P1P2 **clockwise**, is Nextcw. Similarly, the predecessor of P1P2 in the clockwise traversed face F2 is Predcw. Another way to remember this: the next edges share the end of the

current edge (P2), the previous edges share the beginning of the current edge (P1). If the face is traversed clockwise, then we deal with edges Predcw and Nextcw, and if the face is traversed counterclockwise, we deal with edges Predccw and Nextccw.

SUM UP: an edge contains the following informations:

1. **vertices of this edge,** $P1$ **and** $P2$
2. **its left and right faces,** $F1$ **and** $F2$
3. **predecessor (***Predccw***) and successor (***Nextccw***) when traversing its left face (ccw)**
4. **predecessor (***Predcw***) and successor (***Nextcw***) when traversing its right face (cw).**

To spare time and to verify our results, edges and faces can partially been initialized (input 14-15-16 of the program menu), which leads to the following representation:



Some information is already available and can be summed up in the following tables:

Edge Nb	Star Pt	End Pt	Face ccw F1	Face cw F2	Nccw	Pccw	Ncw	Pcw
1	P0	P1	F1					
2	P1	P2						
3	P2	P3						
4	P3	P4						
5	P7	P6	F2					
6	P6	P5						
7	P5	P4						
8	P4	P7						
9	P0	P4	F3					
10	P1	P5	F4					
11	P2	P6	F5					
12	P3	P7	F6					

A face can simply be represented as a pointer to an initial edge, which corresponds in our case to:

Face	Point 1	Point 2	Point 3	Point 4	Start Edge
F1	P0	P1	P2	P3	1
F2	P7	P6	P5	P4	5
F3	P0	P4	P5	P1	9
F4	P1	P5	P6	P2	10
F5	P2	P6	P7	P3	11
F6	P3	P7	P4	P0	12

Exercice 1 (30 minutes)

Fill in (and assert with the teacher) the table of edges.

Exercice 2 (30 minutes): code analysis

Analyze the source code provided and answer to the following questions:

1. How many classes are available? What are their purposes?
2. Comment (using //) the following lines in the *.h* files and see what happens:

```
#ifndef ...
#define ...
...
#endif
```

3. Why does the compiler detect multiple header file inclusions? Therefore, what is the purpose of those lines?
4. What does the main program do?
   (a) How can you add points, edges, and faces using the program? What are the corresponding $C++$ instructions?
   (b) How can you display informations regarding points, edges, and faces? What are the corresponding $C++$ instructions?
   (c) Classes built to handle arrays have a member array that is of $< type > **$. What does this mean? What is the purpose of such structure? How can you use it?
5. Where do you have to insert your own source code?

Exercice 3

Implement the code to complete the whole representation of the cube at the appropriate location within the program. If necessary you can create your own classes, functions, etc.

Exercice 4

If you have correctly represented the cube, implement the code to assert that all the faces are valid, *i.e.* They have 4 edges, each edge is valid (no null pointer), and you can traverse them.