8. Region Growing

Listen

In C++ bietet die STL (tandard Template Library) effiziente Möglichkeiten zum Speichern von Daten, siehe http://www.cplusplus.com/reference/stl/
Im folgenden werden wir eine `std::list` verwenden, die insbesondere das effiziente Hinzufügen und Entfernen einzelner Daten aus einer Sammlung implementiert.

```
1  #include <list>
2  using namespace std;
3  void example(){
4    list<int> mylist;            // create a list for integer values
5    for(int i=0;i<10;i++)
6    {mylist.push_back(i); }      // fills some values into the list
7    while(!mylist.empty()){      // do ... until list is empty
8      int j = mylist.front(); // reads the first element of the list
9      ...
10     mylist.pop_front();        // removes the first element of the list
11   }
12 }
```

Image::TYPE::INT

Da es nicht vorherzusehen ist, wie viele Segmente bei einer Segmentierung entstehen, kann das Labelimage nicht als 8-Bit Grauwertbild gespeichert werden. Statt dessen werden die Labels der Segmente als Integerwerte gespeichert. Daher wird nun TYPE in der Klasse Image wie folgt ergänzt:

```
1  public: enum TYPE{UNDEF=0,BINARY=1,GRAY=2,RGB=3,HSV=4,INT=5};
```

Ergänzen Sie außerdem die Funktion Init() und Save() in der Klasse Image. Zum Speichern eines Labelbildes können Sie z.B. mit dem Modulo-Operator den Wertebereich auf 8-Bit bringen und das Bild dann als normales Grauwertbild speichern und anzeigen - aber dabei entsteht natürlich ein Informationsverlust.
Für den Zugriff auf die Labeldaten, die in der Klasse Image intern als unsigned char gepeichert sind, empfiehlt sich dann ein cast auf int:

```
1  Image labelImage;
2  labelImage.Init(w,h,Image::INT);
3  int* labels = (int*)labelImage.Data;
```

Region Growing

Implementieren Sie nun einen Region Growing Algorithmus für Grauwertbilder

```
1  void RegionGrowingGray(const Image& gray, Image& labelImage,
2       Image& regionImage, int span=0);
```

Dabei ist

`labelImage` vom Typ `INT`, siehe unten

`regionImage` vom Typ `GRAY`. Hier werden die einzelnen Grauwerte der Pixel durch den mittleren Grauwert der jeweilgen Region dargestellt.

`span` ist der Schwellwert für das Ähnlichkeitsmaß.

Hinweise:

- Berechnen Sie zunächst nur das Labelimage.
- Danach können Sie aus dem Labelimage und dem Originalimage die mittleren Grauwerte aller Regionen berechnen und dann im Regionimage speichern.
- Während des Wachsens einer Region müssen Sie Pixel zwischenspeichen, und zwar diejenigen, die Nachbarpixel der gerade wachsenden Region sind und noch nicht verarbeitet wurden. Dazu können Sie die Datenstruktur std::List benutzen.

```
1  #include <list>
2  #include "image.h"
3  using namespace std;
4  void RegionGrowingGray(...){
5    // INIT
6    ...
7    int* labels = (int*)labelImage.Data;
8    int label=0;
9    list<int> listIdx;
10
11   // CREATE LABEL IMAGE
12   for(int x=...) for(int y=...){
13     int idx=...
14     listIdx.push_back(idx); // store current pixel position
15     ...
16     while(!listIdx.empty()){
17     ...
18     idx = listIdx.front();  // get a pixel position of the neighborhood of th
19     ...
20     listIdx.pop_front();    // delete first element
21     ...
22
23   // CALCULATE MEAN VALUE OF EACH SEGMENT
24   ...
25
26   // CREATE REGION IMAGE
27   ...
```