Les objects

<sylvain.berlemont@lrde.epita.fr>

LRDE seminar, February 18, 2003

Table des matières

Points

- Classes: `point1d`, `point2d`, `point3d`
- Represent a position (not a value) in the image.
- Available methods:

```
static std::string name()  // get class's name of
                           // instanciated object

coord nth(const unsigned dim) const  // component
coord& nth(const unsigned dim)       // accessors

coord row() const // 2d point accessors
```

```
coord& row()
coord col() const
coord& col()

bool operator==(const self& p) const // comparaison
bool operator!=(const self& p) const

point2d operator+(const dpoint2d& dp) const // operations
point2d operator-(const dpoint2d& dp) const // on point2d
point2d& operator+=(const dpoint2d& dp)
point2d& operator-=(const dpoint2d& dp)
dpoint2d operator-(const point2d& p) const
point2d operator-() const
```

Delta Point

- Classes: `dpoint1d`, `dpoint2d`, `dpoint3d`
- Represent a movement between 2 points.
- Available methods:

```
static std::string name() // get class's name of
                          // instanciated object

coord nth(const unsigned dim) const  // component
coord& nth(const unsigned dim)       // accessors

bool is_centered(void) const // e.g: dpoint2d(0,0)
```

```
coord row() const // 2d delta point accessors
coord& row()
coord col() const
coord& col()

bool operator==(const self& p) const // comparaison
bool operator!=(const self& p) const

dpoint2d operator+(const dpoint2d& dp) const // operations
dpoint2d operator-() const                   // on dpoint2d
dpoint2d operator-(const dpoint2d& dp) const
dpoint2d& operator+=(const dpoint2d& dp)
dpoint2d& operator-=(const dpoint2d& dp)
```

Iterators

- Classes: `fwd_iter`, `bck_iter`
- Iter upon image point.
- 2 types of iterators: forward and backward iterators.
- Olena iterator is NOT like stl one:

```
image2d<bin> src = load(‘‘lena.pbm’’);
typename image2d<bin>::fwd_iter i(src);
[...]
*i = true;     // ko. ’*i’ doesn’t mean anything.
src[i] = true; // ok. iterator == point.
```

- Available methods:

```
static std::string name() // get class's name of
                          // instanciated object

bool operator==(const point2d& p) const // comparaison
bool operator!=(const point2d& p) const
point2d operator+(const dpoint2d& dp) const // operations
point2d operator-(const dpoint2d& dp) const

point2d cur() const // instead of 'point(it)'

coord row() const // 2d iterator accessors
coord col() const

void operator++() // increment operator
```

Windows

- Classes: `window1d`, `window2d`, `window3d`
- Windows are sets of delta-points.
- Common use:

```
window1d w1;
w1.add(-1).add(0).add(1);
std::cout << "w1 = " << w1 << std::endl;

point1d p(2);
image1d<int_u8> src(10);
for (int i = 0; i < w1.card(); ++i)
```

```
    cout << src[p + w1.dp(i)] << endl;


static std::string name() // get class's name of
                          // instanciated object

window2d& add(const dpoint2d& dp) // add a dpoint2d
window2d& add(coord row, coord col)

bool has(const dpoint& dp) const // check content
unsigned card() const // number of dpoints
bool is_centered() const // acc is 0

dpoint dp(unsigned i) const // accessor

bool operator==(const self& win) const // comparaison
```

- Pre-defined windows:

```
// 4-connexity with dpoint2d(0,0)
inline const window2d& win2d_c4p()

// 4-connexity without dpoint2d(0,0)
inline const window2d& win2d_c4_only()

// same in 8-connexity
inline const window2d& win2d_c8p()
inline const window2d& win2d_c8_only()
```

Weighted Windows

- Classes: `w_window1d`, `w_window2d`, `w_window3d`
- Same as window except that each dpoint is bestowed with a weight
- Common use:

```
w_window2d w;
w.add(-1, -1, 2); // x coord, y coord, weight
cout << ‘‘dpoint: ‘‘ << w.dp(0)
     << ‘‘weight: ‘‘ << w.w(0) << endl;
```

Neighborhood

- Classes: `neighborhood1d`, `neighborhood2d`, `neighborhood3d`
- These objects are like Windows, except they have an additional property:

$$\mathcal{V}_p = \{q\} \text{ means } \begin{cases} p \notin \mathcal{V}_p \\ \forall q \in \mathcal{V}_p \Rightarrow p \in \mathcal{V}_q \end{cases}$$

- Common use:

```
neighborhood1d n1;
n1.add(1);
std::cout << "n1 = " << n1 << std::endl; // N[(-1)(1)]
```

- Available methods: *see Windows methods*

- Pre-defined neighbourhoods:

```
inline const neighborhood2d& neighb_c4() // 4-connexity
inline const neighborhood2d& neighb_c8() // 8-connexity
```