lostream

Thursday, September 10, 2009
3:28 PM

- you've already seen:

  std::ostream& operator<< (std::ostream& s, const T& rhs),

- this works for this usage:

  std::cout << some_obj;

But did you know... same function can be used for writes to a file? And to a binary file?

- Let's consider our PPM image class & PPM file structure:

  PPM: portable pixmap
  Jeff Poskanzer wrote this ca. 90s

magic number

P6	color [r g b]	—	PPM
P5	grayscale [lum]	—	PGM
P4	binary [1/0]	—	PBM

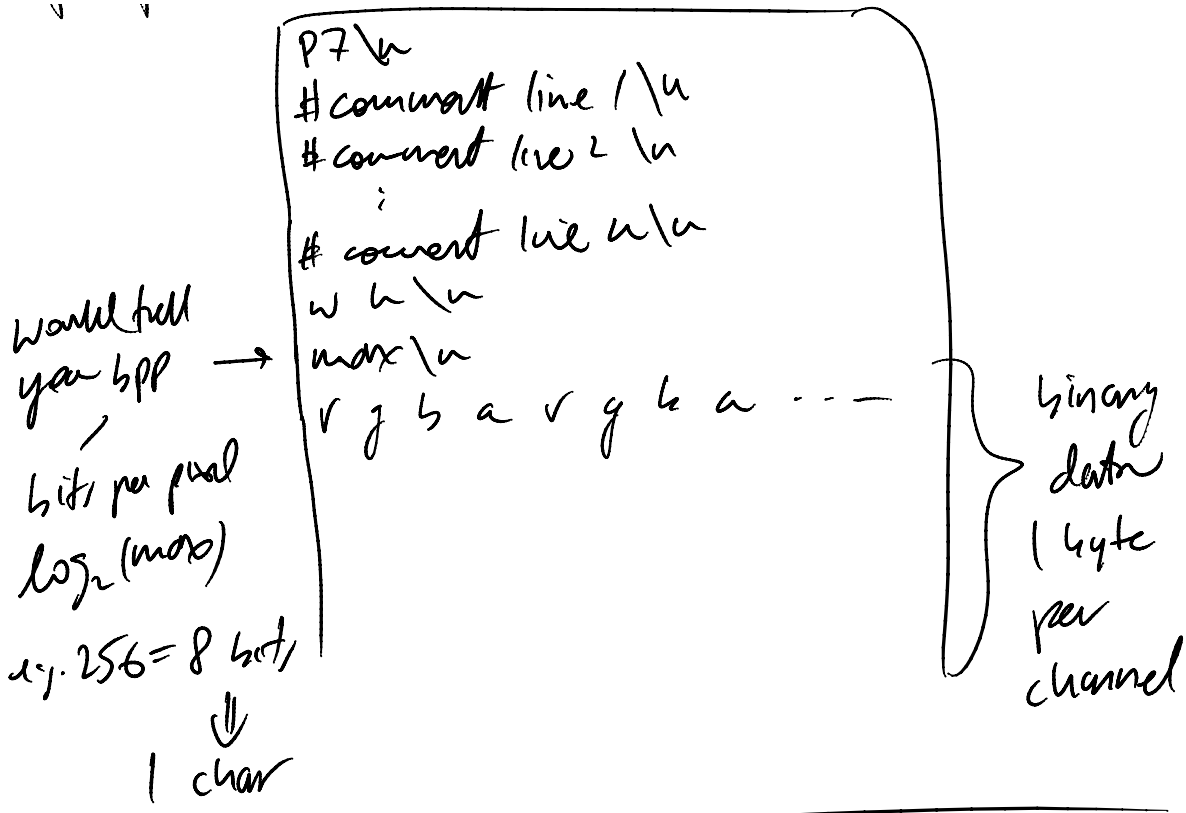
PNM — "any map" (no such file exists)
[pure virtual]
|
would contain code common to all derived classes

P7? doesn't officially exist, have grayscale file format that contains an alpha channel (transparency)

---

file format:

```
P7\n
#comment line 1\n
#comment line 2 \n
```



output

```
std::ostream& operator << (std::ostream& s, const Pnm& rhs)
{
    s << 'P' << rhs.ptype << std::endl;   // "\n" ? ok.
    s << "# stored by pnm class." << std::endl;
    s << rhs.w << " " << rhs.h << "\n";
    s << rhs.maxv << std::endl;
    s.write(rhs.img, rhs.bpp * rhs.w * rhs.height);
    return s;
}
```

binary file write

the 1D image array

file should be open in "binary" mode

usage:

```
std::ofstream ofs;
```

output file

```
std::ofstream ofs;

ofs.open("filename", std::ios::out | std::ios::binary);
ofs << p[i];
ofs.close();
```

logical or

- reading in (parsing) a bit more complicated

std::istream& operator>> (std::istream& s, Ppm& rhs)

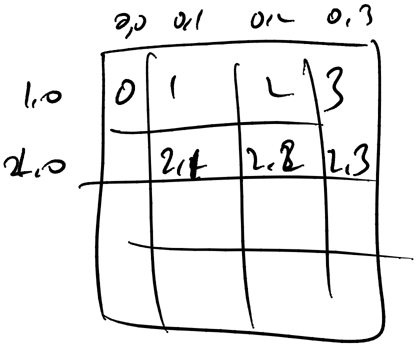
↳ doesn't have to be ifstream because ifstream is a istream

```
{
  char c;
                              (expect)
  s=s.get(); // read one character 'p'
  if (s.eof()) // end of file reached

  s >> rhs.ptype >> std::ws;
           ↑              ↑
        an int        white space ('\n')
  (and s obliges, converts '7' to an int)

  // check for EOF
  switch (rhs.ptype) {
      // set rhs.bpp
  }
                 ⟶ lookahead, don't consume
  while ((c = s.peek()) == '#') do { s.get(c); }
                                while (c != '\n');

  s >> rhs.w >> rhs.h >> std::ws;
            ↑
     eats white space between w h

  s >> rhs.maxo >> std::ws;
  rhs.img = new char [rhs.bpp * w * h];
                              |
                          bytes per pixel
  s.read (rhs.img,
```

s.read((char*)img,
bpp * w * h); — quick way to read whole thing in
per pixel
return s;
}

Another way:

```
for(int i=0; i<h; i++)
  for(int j=0; j<w; j++)
    s >> rhs.img[i*w + j]
```



this allows two type of usage:

ifstream ifs
ifs.open()
ifs >> img >> some ~~other~~ thing to read
ifs >> "

- suppose I have two layers
  1 color & 1 grey
  (color plate) (matte)
  (bitmask)

composite (used in compositing, e.g. green screening / blue screening) chroma keying)

how to merge:

```
    ppm  c("newcolor.ppm"); // color
    ppm  g("newgrey.ppm");
    ppm  pic("newcombined.ppm");
                                        ______ if ifs.good()
  pic.merge(c,g);
  ofs.open(pic.fname().c_str(), out|binary)
              std::string  char*

  ofs << pic;
  ofs.close()
}
```

```
void ppm::merge(ppm& c, ppm& g)
{
  //error checking
  if ((c.w != g.w) || (c.h != g.h)) return;
  if ((c.ptype != 6) || (g.ptype != 5)) return;

  ptype = 7;    //means (*this).ptype
  w = c.w;
  h = c.h;
  bpp = 4;
  max = c.cmax;
  if (img) { delete [] img; }
  img = new char [bpp * w * h];
  ...                            // ptrs. to arrays
```

```
// char *p7, *p6, *p5; // ptrs. to arrays
p7 = img; p6 = c.img; p5 = g.img;
for (int i=0; i < w*h; i++) {
    *p7++ = *p6++; // r
    *p7++ = *p6++; // g
    *p7++ = *p6++; // b
    *p7++ = *p5++  // a
}
```

↑ Like iterators