Subject: Basic character set analyzer
Posted by cbpporter on Sat, 13 Dec 2008 10:26:41 GMT
View Forum Message <> Reply to Message

Before starting to write a text output method capable of font substitution, using a little program to see what fonts contain which characters is a good idea (Mirek's idea).

I've already gotten a lot of valuable information from this little program, so if anybody want to give it a try, run it and post it's result here (or as PM if you don't want to fill up the space here) so I can tell if my assumptions hold out for different versions of Linux, that would be great. I would like to see results both from people who just installed a normal Linux, never bothering to look over the font list and from people who have a localized version of Linux or who manually installed a font to be able to use non English characters.

The program covers code ranges from Basic Latin to Arabic. I did not go any farther yet, because clearly a better interface and way to present information is needed (I'm thinking tables, ranges and percents vs. current character list). But this will be enough to verify my assumptions and write a basic method which will handle for now only the above code ranges.

PS:
To make this work, you are going to have to add this to Draw/Draw.h in FontInfo declaration:

```
bool       HasChar(int codePoint);
bool       HasCharRange(int startCp, int endCp);
bool       CharRangeEmpty(int startCp, int endCp);
```

and this to Draw/DrawText.cpp:

```
bool FontInfo::HasChar(int codePoint)
{
 return XftCharExists(Xdisplay, ptr->xftfont, codePoint);
}

bool FontInfo::HasCharRange(int startCp, int endCp)
{
 for (int i = startCp; i <= endCp; i++)
  if (!XftCharExists(Xdisplay, ptr->xftfont, i))
      return false;
 return true;
}

bool FontInfo::CharRangeEmpty(int startCp, int endCp)
{
 for (int i = startCp; i <= endCp; i++)
  if (XftCharExists(Xdisplay, ptr->xftfont, i))
      return false;
 return true;
}
```

These methods are not final, so add them only for the purpose of running this test.

File Attachments

1) FontScript.tar.gz, downloaded 562 times

Subject: Re: Basic character set analyzer
Posted by cbpporter on Wed, 17 Dec 2008 10:21:48 GMT
View Forum Message <> Reply to Message

Quote:
I've continued experimenting and reached an interesting conclusion: X does some very basic character substitution and composition for Latin characters.

I tested pretty much every font that doesn't have Latin-1 Supplement support and it seems that if the given font has basic Latin alphabetic characters, it will manage to draw also those from the supplement by doing on the fly composition. The results are pretty good. I think we can skip substitution for the supplement if alphabetic characters are available. If alphabet is not available, we need to substitute those characters and use probably the same method you used for replacing Latin characters for Font::COMPOSED.

Also, X tries to compose missing characters from Latin Extended-A. Some fonts handle this OK, but most will either fail, or draw a rectangular outline and superimpose the root character of the composition. E.G. if I have a very strange "L" with dots on it's right, I will get a box and a normal L. In this case I don't know which is better. Using substitution will result in a correct character, but quite likely the typeface will be different. Using X scheme, we get the same type face, but the result is ugly.

This is what I was just about to post this when I realized: Maybe X is not the one who is doing the composition. Maybe It's U++ with the mechanism I mentioned earlier! Am I right? Or is the composition used only under Windows. I couldn't figure it out by testing under other application, because all feature under a form or other font substitution. Does anybody know a very primitive plain X editor which can choose it's font. Or maybe a font exploration tool for Linux?

Subject: Re: Basic character set analyzer
Posted by cbpporter on Wed, 17 Dec 2008 16:18:34 GMT
View Forum Message <> Reply to Message

I found the code responsible for composition, and it seems only code points 0x0100 to 0x017F are subject to U++ composition. So either X11 does it's own composition for Latin-1 Supplement, or Xft API is lying to us about which characters are available (or some other reason).

I'll try to determine more. But for now, first step is going to be to make characters available when basic Latin is missing in font. I think this is a good idea. Even if you use Dingbats or some other specialized fonts, I think it would be useful to be able to print basic Latin characters without having to use two explicit fonts. I'll use StdFont as a basic Latin fallback, since this font will always contain the needed characters.

I also noticed that using a lot of fonts slows down rendering to a crawl. I'll have to look over the code to see if some caching can be done or some bottleneck avoided, but basically this means that we must keep the substitution pool as small as possible.

PS: How was the current composition behavior established? How did you determine that you need to draw the little line at an offset of font.GetHeight() / 13 for example. Did you find some reference material, or was it experimental and you went with what looked good.

Subject: Re: Basic character set analyzer
Posted by mirek on Thu, 18 Dec 2008 10:34:55 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Wed, 17 December 2008 11:18I found the code responsible for composition, and it seems only code points 0x0100 to 0x017F are subject to U++ composition.

Yep, with the goal of covering most of europe It definitely covers czech language and it is in fact very nice to be able to use all fonts for czech

I am tuned for the next results. Knowing how all these things really work would be invaluable.

In the same time, I think it is reasonable to expect that very soon we will have to go down one level and use FontConfig and FreeType directly (not Xft). The pressure is coming from software rendering development. We want U++ that is able to Draw without X11....

I also noticed that using a lot of fonts slows down rendering to a crawl. I'll have to look over the code to see if some caching can be done or some bottleneck avoided, but basically this means that we must keep the substitution pool as small as possible.

This is quite logical, is not it? I believe it is not even U++ problem. Xft has to cache thing too.

Quote:
PS: How was the current composition behavior established? How did you determine that you need to draw the little line at an offset of font.GetHeight() / 13 for example. Did you find some reference material, or was it experimental and you went with what looked good.

Experimental.

Have taken the default sans-serif font in Linux and tweaked until looked acceptable. It is heurestics heavy

Mirek

Subject: Re: Basic character set analyzer
Posted by cbpporter on Sat, 24 Jan 2009 10:02:43 GMT
View Forum Message <> Reply to Message

Hi!

It's been a long time since I last posted. But I've been very busy. In December I decided to follow a rather interesting opportunity. Long story short: I have a new job now in the domain of industrial printers!

But things are starting to calm down now and I thought I'd tackle my old nemesis: font rendering.

Here are a couple of screenshots. The first is with default U++ rendering, the second is with my modifications:

It seems I am falling into the common font substitution pitfall: you can render anything but it will look like ass. Especially on an outline font like this one. But anyway, the second result is still a lot more readable and I guess it would be preferred for practical circumstances.

File Attachments
1) snapshot3.png, downloaded 1292 times
2) snapshot2.png, downloaded 1106 times

Subject: Re: Basic character set analyzer
Posted by mirek on Sat, 24 Jan 2009 13:00:21 GMT
View Forum Message <> Reply to Message

Well, good news is that recent Painter development introduced me much closer to actual font rendering... I guess that in a short while, we will have something reasonable for font substitution.

Mirek

Subject: Re: Basic character set analyzer
Posted by cbpporter on Sun, 25 Jan 2009 09:14:26 GMT
View Forum Message <> Reply to Message

I experimented a lot today with witting a new diacritics composer. I thought since it is so hard to get the problematic fonts look right, I would write something that would work fine for non-problematic fonts and then try to figure out where the difference is. I obtained a very basic version which draws diacritics by composition, even if precomposed character are available in font. I used step by step comparison to try to get the proportions right.

And even thought I succeed in most cases, it is very hard to determine a general formula bases only on character height, ascent and descent. I will need base height and x-height also. Looking over XGlyphInfo it seems that it is possible to get the exact height and width of a given character.

Also, using the macron character will yield uniformly horrible results, so I replaced it with a basic '-'.

Subject: Re: Basic character set analyzer
Posted by mirek on Sun, 25 Jan 2009 09:56:24 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Sun, 25 January 2009 04:14I experimented a lot today with witting a new diacritics composer. I thought since it is so hard to get the problematic fonts look right, I would write something that would work fine for non-problematic fonts and then try to figure out where the difference is. I obtained a very basic version which draws diacritics by composition, even if precomposed character are available in font. I used step by step comparison to try to get the proportions right.

And even thought I succeed in most cases, it is very hard to determine a general formula bases only on character height, ascent and descent. I will need base height and x-height also. Looking over XGlyphInfo it seems that it is possible to get the exact height and width of a given character.

Also, using the macron character will yield uniformly horrible results, so I replaced it with a basic '-'.

I wonder how is this effort different from existing code?

Quote:

Yes. Wiki.

Mirek

Subject: Re: Basic character set analyzer
Posted by cbpporter on Wed, 28 Jan 2009 18:53:12 GMT
View Forum Message <> Reply to Message

luzr wrote on Sun, 25 January 2009 11:56
I wonder how is this effort different from existing code?

Well I hope it is going to be different in at least one detail:

It is going to work now (as soon as I'm ready). I've been nagging forever about the horrible font support under Unix, yet the situation remained the same. Don't take this the wrong way: I am well aware how busy people very providing genuine improvements all over U++, but this domain has largely been ignored. I understand that this is not a focus area right now, but I don't have time to wait until it will become one. And while Painter might help here, I do need a version which works with plain X, because I have no intention to ship with AGG.

I believe I am very close to a general, powerful and most importantly good looking mechanism. Here is a sample screenshot:

As you can see, the manual placement of diacritics (rendered in red) is almost as good as the native one. There are still some bugs left, but the most important part is that the mechanism is general. It is based on precise bounding boxes, so I can use for diacritics any renderable character, not just special ones. This opens the way to full composition support with very little extra work.

Quote:
Yes. Wiki.

You know that you've been staring to much at this site when the disappearance of a link bothers you . Anyway, the wiki was in a bad shape and I can't say I'm going to miss it. It wasn't really that useful anyway, since we don't really have that kind of a community which favors wikis.

File Attachments

1) snapshot4.png, downloaded 1276 times

Subject: Re: Basic character set analyzer
Posted by mirek on Fri, 30 Jan 2009 07:53:37 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Wed, 28 January 2009 13:53
As you can see, the manual placement of diacritics (rendered in red) is almost as good as the native one. There are still some bugs left, but the most important part is that the mechanism is general. It is based on precise bounding boxes, so I can use for diacritics any renderable character, not just special ones. This opens the way to full composition support with very little extra work.

I still do not get it. I thought we are already doing this in Draw/ComposeText.cpp.

I guess I will have to wait for the actual code...

Mirek

Subject: Re: Basic character set analyzer
Posted by cbpporter on Fri, 30 Jan 2009 08:15:37 GMT
View Forum Message <> Reply to Message

luzr wrote on Fri, 30 January 2009 09:53
I still do not get it. I thought we are already doing this in Draw/ComposeText.cpp.

I guess I will have to wait for the actual code...

Yes, we are doing it, but we could just as well not do it. A lot of fonts have these characters, so the code is not used.

And in places where the code is used, it looks very bad. I posted a screenshot a while ago with exactly how the text looks:

This is completely unreadable. I don't need all the characters that are above, but I don't like half baked solutions, so I would like them all to work.

The problem is that even though rendering is OK for Latin fonts, it is horrible for non-Latin fonts and I must use almost non-Latin fonts exclusively. Plus I need composition for non-Latin characters, so I might as well do it first for Latin ones, where the implementation is several orders of magnitude easier.

Subject: Re: Basic character set analyzer
Posted by mirek on Fri, 30 Jan 2009 08:28:07 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Fri, 30 January 2009 03:15luzr wrote on Fri, 30 January 2009 09:53
I still do not get it. I thought we are already doing this in Draw/ComposeText.cpp.

I guess I will have to wait for the actual code...

Yes, we are doing it, but we could just as well not do it. A lot of fonts have these characters, so the code is not used.

And in places where the code is used, it looks very bad. I posted a screenshot a while ago with exactly how the text looks:

This is completely unreadable. I don't need all the characters that are above, but I don't like half baked solutions, so I would like them all to work.

Good, getting somewhere.

I believe the problem with the above font is that it does not have diacritics characters defined.

What is your solution to the problem?

Quote:
The problem is that even though rendering is OK for Latin fonts, it is horrible for non-Latin fonts and I must use almost non-Latin fonts exclusively. Plus I need composition for non-Latin characters, so I might as well do it first for Latin ones, where the implementation is several orders of magnitude easier.

I wonder how do you plan to do composition for non-Latin characters if basic glyphs are not defined in the font?

Hm, maybe it is just terminology issue - by composition I mean creating a new glyph by composing two other glyphs (basic latin character + diacritics glyph) from the same font.

Mirek

Subject: Re: Basic character set analyzer
Posted by cbpporter on Fri, 30 Jan 2009 08:59:05 GMT
View Forum Message <> Reply to Message

luzr wrote on Fri, 30 January 2009 10:28
Good, getting somewhere.

I believe the problem with the above font is that it does not have diacritics characters defined.

What is your solution to the problem?

That's exactly the problem. I'm using two way substitution, replacing both base character and diacritic when necessary. Diacritics look similar enough across fonts so no problems here with substitution. For base char, I'm using StdFont. This way at least characters substituted across different fonts will at least look the same, even though the might differ visually from current font. But this is not a big problem for Latin, because most font either have all characters from a range or have none. There are some who have an arbitrary subset of those characters, but I'm guessing that the subset is biased toward an existing language and since people will probably use that font to render that given language, we shouldn't have many cases where substitution result in ugly text.

Quote:
Hm, maybe it is just terminology issue - by composition I mean creating a new glyph by composing two other glyphs (basic latin character + diacritics glyph) from the same font.

That's for Latin composition (which I'm implementing right now). For non-Latin composition, I create a new glyph based on a non-Latin little drawing and combine it with another one (which could be considered a diacritic).

The reasons why I'm not altering your code and witting a new one are:
1. I want to have full composition with arbitrary bases and diacritics to handle all Unicode composition characters. I can place a '-' on something, but I can also place any printable character.
2. I am implementing these methods as functions that take a Draw object as their first parameter so that I can use the code without having to integrate it into Draw. While some minor additions are necessary to Draw (a method to determine if font has char and one to determine the exact bounding box of a character), getting these accepted is going to be easier than a full blown Unicode composition engine with heavy bias toward Latin and CJK. And BTW, I'm still completely against Utf32, so it's safer this way .

Subject: Re: Basic character set analyzer
Posted by cbpporter on Sat, 31 Jan 2009 12:22:08 GMT
View Forum Message <> Reply to Message

I've made great progress today and all fonts are capable of displaying almost all Latin characters and compositions. There is one last problem and this is a big one.

added when I interrogate the font for the top and bottom of '`', I will get something in the vicinity of 2 for top (it doesn't start exactly at the top of the line) and something in the vicinity of 5 for bottom (these numbers are just for example). I need this info, rather then ascent/height. And it works for most fonts. But there are some fonts which will return 0 for the top and the height of the font for bottom. So instead of getting (2, 5), I will get (0, 18) and this ruins all computations and causes the diacritic to be severely misplaced. This is not the correct value for the API calls that I have made, and I think it is the fault of the font designer, who did not alter the vertical extent information of the character and just went with the default font height.
 I hope I was successful in explaining this issue.

I don't know how I could fix this, safe for rendering the character in a white bitmap buffer, and shrinking the bitmap line by line until I have reached the minimal height and caching the result. I'm afraid such a method will be slow, and anyway I would rather not take such extreme measures.

There is one other solution: define one global font for diacritics and draw all diacritics with that

base, but the diacritic is going to look constant. I think this is a reasonable compromise.

What do you think?

Subject: Re: Basic character set analyzer
Posted by mr_ped on Sat, 31 Jan 2009 13:18:23 GMT

View Forum Message <> Reply to Message

If the behavior of returning [0,font_height] is consistent, you can make "some" solution to trigger only in this specific case, otherwise you can go with your current "correct" version?

How the "some" solution should look... If you will stick with above mentioned rule, maybe a global font fallback sounds sufficiently.

Subject: Re: Basic character set analyzer
Posted by cbpporter on Sat, 31 Jan 2009 23:53:49 GMT
View Forum Message <> Reply to Message

I decided to follow the idea with a predetermined list of fonts for diacritics (unfortunately there isn't one that contains all the characters, so I'll have to use more to cover a wider range). For now I'll simplify the issue, by using one single hand-picked font. This should make things a lot easier and render most characters that are likely to appear.

I also combined this method with pin point accurate character positioning (but vulnerable to the problem when the API returns wrong text extents) and now I can combine characters at will. Here is more complicated and actually impossible result of a combination, but which demonstrates the precision of the method:

File Attachments
1) snapshot5.png, downloaded 994 times

Subject: Re: Basic character set analyzer
Posted by cbpporter on Sun, 01 Feb 2009 15:03:51 GMT
View Forum Message <> Reply to Message

I gotten to a point where I could render OK about 80% of the fonts. But after hours of trial an error I couldn't figure out why the last 20% were rendered wrong.

So I decided that current method is not powerful enough, scratched everything and started almost from zero. Good news is that I found a new method and this one works extremely well. This method offers great results for 100% of the fonts (at least the ones on my system). The bad news is that old tables are no longer good and I have to update all tables which will take some while.

I attached a screenshot with the first two characters from Latin Extended A. I believe it is not practical to try to obtain better results, especially since I think it looks great.

File Attachments
1) snapshot6.png, downloaded 474 times

Subject: Re: Basic character set analyzer
Posted by mirek on Mon, 02 Feb 2009 08:40:49 GMT
View Forum Message <> Reply to Message

Well, my original plan was to perform composition if both glyphs are available in the font (current algorithm), then look into other fonts to get the complete required glyph...

Mirek

Subject: Re: Basic character set analyzer
Posted by cbpporter on Mon, 02 Feb 2009 10:19:06 GMT
View Forum Message <> Reply to Message

luzr wrote on Mon, 02 February 2009 10:40Well, my original plan was to perform composition if both glyphs are available in the font (current algorithm), then look into other fonts to get the complete required glyph...

Mirek
Yes, I thought about that but I tried with this method because this way at least the basic character will look the same way as the font. Maybe it's worth a shot to try it your way also.

But anyway, we should get to the modifications in Draw to handle these algorithms. Nothing must be changed, but some methods must be added.

The first one is HasChar from the first post in this thread (I no longer need HasCharRange and CharRangeIsEmpty). So I propose either add HasChar, or if you have a better method to do it (maybe one for Win also), I'm opened for suggestions.

PS: HasChar seems pretty fast. I run it for every font on application startup for code ranges from zero to the end of Arabic range, and I didn't notice any slowdown, so I guess it is fast enough to render all the text that can appear on screen at once. At least one call for HasChar must be made for every character that will be printed, so maybe later we can cache results somewhere in home folder.

Subject: Re: Basic character set analyzer
Posted by mirek on Sat, 07 Feb 2009 13:53:01 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Mon, 02 February 2009 05:19
PS: HasChar seems pretty fast. I run it for every font on application startup for code ranges from zero to the end of Arabic range, and I didn't notice any slowdown, so I guess it is fast enough to render all the text that can appear on screen at once. At least one call for HasChar must be made for every character that will be printed, so maybe later we can cache results somewhere in home folder.

I apologize for delay; HasChar is now officially in Draw, implemented both for Win32 and X11.

Well, thinking about the issue - do you have any idea how to determine which fonts search for replacements first?

I mean, for missing char in Arial, I would check some sans-serif font first rather than trying e.g. Times New Roman..

Mirek

Subject: Re: Basic character set analyzer
Posted by cbpporter on Sun, 08 Feb 2009 11:45:34 GMT
View Forum Message <> Reply to Message

---

luzr wrote on Sat, 07 February 2009 15:53
I apologize for delay; HasChar is now officially in Draw, implemented both for Win32 and X11.

Great! I'll check out the win version too, but right now all my efforts focus on X.

Quote:
Well, thinking about the issue - do you have any idea how to determine which fonts search for replacements first?

I mean, for missing char in Arial, I would check some sans-serif font first rather than trying e.g. Times New Roman..

Mirek
I've given this issues a lot of thought, but I don't think there is a good way to do this.

This is why I chose composition rather than substitution.

So basically we have the first 3 Unicode sections which handle Latin. All characters are letters, punctuation and composites. I have basically two cases when a character is missing:
1. Font still has basic Latin characters. In this case I apply composition. I managed to fine tune diacritic placement and results are generally better than when replacing whole character from a different font., but for some size, it will look uglly.
2. Font doesn't have basic Latin. These font are quite rare, and in such cases I use substitution. But these fonts won't really have any Latin support, so I end up using substitution for every character, so font still looks consistent.

But for further Unicode sections the situation changes. These characters start to look less and less Latin, so the problem of substituting with a similar font starts to diminish. I mean, using Arial to draw an 'a' next to a strange fork like character might be a better choice than using Times, but it wont really make a difference. So here I will also use a single font for substitution.

So basically I divide the Unicode range in sub ranges, each containing roughly a single script. And I'll have a vector of vectors of fonts, with first vector indexed by the script number.
Algorithm is in pseudo code:

```
get scriptindex;
if (sciptindex favors composition)
  compose characters
else
  traverse font list until character found; draw box if not found
```
This approach works right now, but since it only handles a small number of scripts, I can't really say yet if it will be enough for real needs of rendering real internationalized texts. Only time will tell.

I attached a screenshot showing the results when using one of the later Latin ranges. Cyan characters are substituted, and also give a nice visual of how average fonts support such characters. Also illustrates problems with using fonts that look ugly one near the other.

Also notice the two lines in Arabic on the right. This is very early support for RTL languages. I give the text in normal left to right order, containing "abc", some Arabic characters, and "def". The text output method detects that the Arabic should be right to left, and renders it as such. First line does plain rendering, second uses specific Arabic rendering rules. While the two texts contain the same characters, the first is wrong and only the second is a correct way to render the characters from the first string.

PS: I can't read Arabic, so I have no idea what I wrote there. I hope I didn't manage to find randomly a swear word or anything like that . I we have users who can read Arabic, I would appreciate some help in this area.

File Attachments

1) snapshot7.png, downloaded 478 times

Subject: Re: Basic character set analyzer
Posted by cbpporter on Sun, 08 Feb 2009 11:55:17 GMT

View Forum Message <> Reply to Message

A little info about the second and final change I need to Draw.

I've added some fields to CharMetrics and some methods to FontInfo. These changes are ad-hoc hacks, and I really need a better integrated solution with proper names.

```
struct CharMetrics : Moveable<CharMetrics> {
  int  width;
  int  lspc;
  int  rspc;
  int  y;
  int  height;
  int  x;
  int  ew;

  bool operator==(const CharMetrics& b) const
      { return width == b.width && lspc == b.lspc && rspc == b.rspc; }
```

```
 };
int       GetY(int c) const                  { return GetCM(c).y; }
 int       GetX(int c) const                  { return GetCM(c).x; }
 int       GetH(int c) const                  { return GetCM(c).height; }
 int       GetW(int c) const                  { return GetCM(c).ew; }
```
These methods are used to determine the exact glyph bounding box, using this method:
```
inline Rect GetCharRect(int x, int y, int buff, const FontInfo& fi)
{
 return Rect(x - fi.GetX(buff), y - fi.GetY(buff) + fi.GetAscent(), x - fi.GetX(buff) + fi.GetW(buff), y -
fi.GetY(buff) + fi.GetAscent() + fi.GetH(buff));
}
```
Also:
```
void FontInfo::Data::GetMetrics(CharMetrics *t, int from, int count)
{
 DrawLock __;
 LTIMING("GetMetrics");
 LLOG("GetMetrics " << font << " " << from << ", " << count);
 if(xftfont) {
  for(int i = 0; i < count; i++) {
   LTIMING("XftTextExtents16");
   wchar h = from + i;
   XGlyphInfo info;
   XftTextExtents16(Xdisplay, xftfont0, &h, 1, &info);
   t[i].width = info.xOff;
   t[i].lspc = -info.x;
   t[i].rspc = info.xOff - info.width + info.x;
   t[i].y = info.y;
   t[i].height = info.height;
   t[i].x = info.x;
   t[i].ew = info.width;
  }
 }
}
```
I don't know if we should cache this.

I'm posting this to show what I need, but before we apply it I really need to clean this up.

Subject: Re: Basic character set analyzer
Posted by mirek on Mon, 09 Feb 2009 07:42:10 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Sun, 08 February 2009 06:45luzr wrote on Sat, 07 February 2009 15:53
I mean, for missing char in Arial, I would check some sans-serif font first rather than trying e.g. Times New Roman..

I've given this issues a lot of thought, but I don't think there is a good way to do this.

Well, first possibility is fixed table of substitutions (the number of widely used fonts is quite small IMO).

Second is PANOSE:

http://en.wikipedia.org/wiki/PANOSE

http://msdn.microsoft.com/en-us/library/ms534014(VS.85).aspx

(-> Win32 provides PANOSE number)

- not sure whether FreeType supports PANOSE... Also, another question is whether fonts actually support it

Third: Perform matching based on visual appearance. That is a bit though, but still possible.

Can be based on raster data processing or even curve processing....

E.g. examining "I" character raster, it should be quite easy to identify serifs, width of central line (and perhaps some other aspects I do not see yet...). By comparing width of 'l' and 'm', you should be able to tell the font is monospace (that is, if this is not stored as attribute somewhere in the font).

Mirek

Subject: Re: Basic character set analyzer
Posted by mirek on Mon, 09 Feb 2009 07:47:41 GMT

View Forum Message <> Reply to Message

cbpporter wrote on Sun, 08 February 2009 06:55A little info about the second and final change I need to Draw.

I've added some fields to CharMetrics and some methods to FontInfo. These changes are ad-hoc hacks, and I really need a better integrated solution with proper names.
[code]struct CharMetrics : Moveable<CharMetrics> {
```
  int  width;
  int  lspc;
  int  rspc;
  int  y;
  int  height;
  int  x;
  int  ew;

  bool operator==(const CharMetrics& b) const
       { return width == b.width && lspc == b.lspc && rspc == b.rspc; }
```

Not really happy about it -> it makes CharMetrics too long.

I think we should read this directly, not to cache this.

I think that, at the end, we should in fact cache 'direct translation'. At the heart of font system there should be a function like:

```
struct RenderGlyph {
    int chr;
    Font fnt;
    int aux_chr; // == 0 -> no aux glyph
    Font aux_fnt;
    int16 aux_x, aux_y;
};


RenderGlyph GetRenderGlyph(int chr, Font fnt);
```

and we should cache it at this phase. We can easily afford to cache tens of thousands of such pairs.

Mirek

Subject: Re: Basic character set analyzer
Posted by cbpporter on Mon, 09 Feb 2009 17:38:43 GMT
View Forum Message <> Reply to Message

Quote:Not really happy about it -> it makes CharMetrics too long.

I think we should read this directly, not to cache this.
OK. How about adding this to FontInfo::Data:

```
Rect FontInfo::Data::GetCharBoundsOffset(int c)
{
 Rect r;
 XGlyphInfo info;
 unsigned int h = static_cast<unsigned int>(c);
 XftTextExtents32(Xdisplay, xftfont0, &h, 1, &info);
 r.left = -info.x;
 r.top = -info.y + ascent;
 r.right = -info.x + info.width;
 r.bottom = -info.y + ascent + info.height;
 return r;
}
```

and this to FontInfo:
```
Rect       GetCharBoundsOffset(int c) const   { return ptr->GetCharBoundsOffset(c); }
```

It is much cleaner this way, plus I can rewrite:

```
inline Rect GetCharRect(int x, int y, int buff, const FontInfo& fi)
{
 return fi.GetCharBoundsOffset(buff).Offseted(x, y);
}
```