Graphismes 2D haute fidélité pour C++

Maxim Shemanarev

Anti-Grain Geometry (AGG) est une bibliothèque graphique 2D libre et gratuite proposant un support pour l'exactitude des éléments de pixels et l'anticrénelage sur plusieurs niveaux. Écrite en C++, AGG est une bibliothèque de rendu capable de créer des images tramées en mémoire à partir de certains types de représentations vectorielles.

AGG se révèle très pratique dans les applications nécessitant vitesse et graphismes 2D haut de gamme, comme par exemple, les applications de systèmes d'informations géographiques ou de cartographie, les interfaces utilisateur graphiques, les tableaux et les diagrammes, la conception et la fabrication assistée par ordinateur, et ainsi de suite. Par ailleurs, AGG ne dépend d'aucune plateforme, tout en restant légère, compacte et robuste. Autrement dit, cette bibliothèque convient parfaitement aux systèmes embarqués et aux dispositifs mobiles.

Les fonctionnalités clés d'AGG sont les suivantes :

- *r*endu de polygones rapide et de très bonne qualité, avec 256 niveaux d'anticrénelage. Application possible de remplissage non nul et même de règles plus étranges encore.
- Vecteur personnalisable et architecture pipeline de lignes.
- Pentes arbitraires et ombrage Gouraud.
- Transformations affines d'images grâce à différentes sortes d'interpolations, de l'interpolation bilinéaire aux interpolations haut de gamme de Lancosz et de Blackman.
- Remplissage des modèles au moyen de transformations affines et de perspectives arbitraires (peut être utilisé pour le procédé de textures).
- Transformations de perspectives et transformations bilinéaires des données sur les vecteurs et les images.
- Générateur de caractères par traits au moyen de différents types de jonctions et de débits de lignes.
- Générateur de lignes en pointillé.
- Marqueurs, comme les flèches haut et bas.
- Découpeur rapide de polygones vectoriels en rectangle.
- Découpage minimal en plusieurs régions rectangulaires.

Sur l'auteur :

Pour contacter l'auteur :
mcseem@antigrain.com
Site officiel du projet : *http://www.antigrain.com*

- Masque alpha.
- Algorithme rapide de lignes anticrénelées.
- Images arbitraires en tant que modèles de lignes.
- Rendu en canaux de couleur séparés.
- Opérations booléennes de polygones (and, or, xor, sub) selon l'algorithme *General Polygon Clipper* d'Alan Murta.
- Algèbre booléenne de perspectives. Permet de réaliser des opérations booléennes sur des formes tramées mises en perspective. Fonctionne 5 à 10 fois plus vite, en moyenne, que l'algorithme GPC.
- Support textuel utilisant la bibliothèque FreeType (*http://www.freetype.org*) ainsi que l'interface de programmation Windows (GetGlyphOutline()).
- Transformations non-linéaires arbitraires.

AGG a été jusqu'ici compilée et testée positivement sur les platef-ormes suivantes :

- Microsoft Visual C++ 5/6/7, Intel C++ 6, GNU C++, version 2.96 à 3.4.0, et les compilateurs Metrowerks CodeWarrior 8.3.
- Windows (95/98/NT4/2000/XP/2003).
- Linux.
- SunOS.
- SGI IRIX64.
- MacOS 9, MacOS X.
- QNX.
- BeOS.
- AmigaOS.

AGG ne comporte aucun modèle de rendu prédéfini. Il s'agit plus d'un *outil permettant de créer d'autres outils*. L'objectif initial de cette bibliothèque a inspiré son architecture et sa philosophie : proposer un ensemble d'outils 2D rassemblés sous un concept commun. L'utilisateur de cette bibliothèque est le seul à pouvoir définir l'architecture résultante et le modèle de rendu. Bien qu'ouverte et flexible, AGG n'est toutefois pas aussi simple à utiliser que des bibliothèques comme GDI+ ou Quartz.

À l'heure actuelle, AGG ne supporte qu'un modèle par chemins, ce qui ne facilite certes pas le rendu de graphismes depuis un format Macromedia Flash. Dans la mesure où l'approche de Flash repose sur les contours, il est possible de rendre une scène multicolore en un seul essai. Si cette méthode reste la meilleure dans certains cas, elle s'avère généralement restrictive et plus difficile à utiliser. En d'autres mots, AGG est plus centrée sur les graphismes vectoriels redimensionnables que ne l'est Flash. Nous avons exposé dans la Figure 1 l'architecture classique d'un moteur de rendu basé sur AGG.

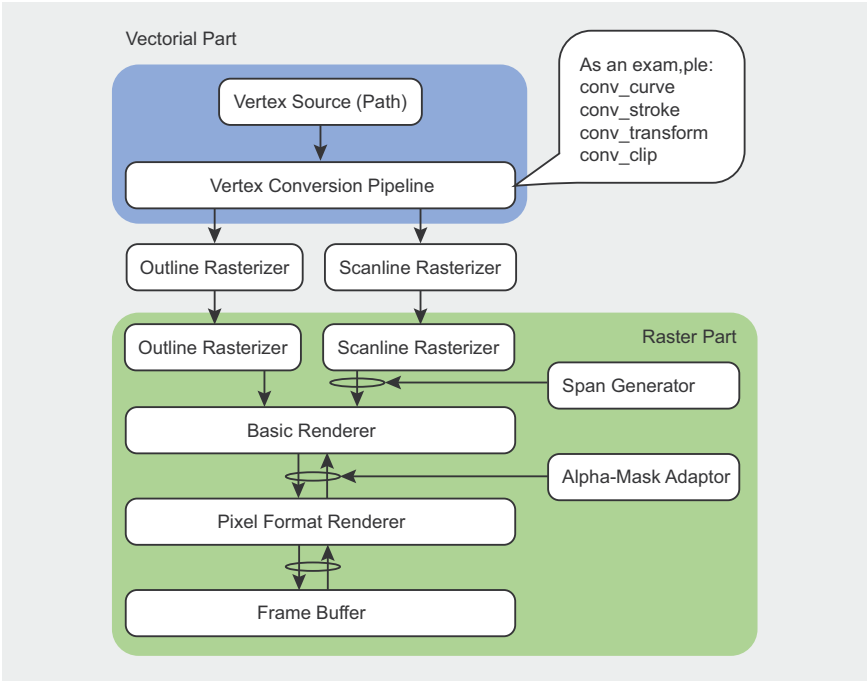

Figure 1. *Architecture classique d'un moteur de rendu basé sur AGG*

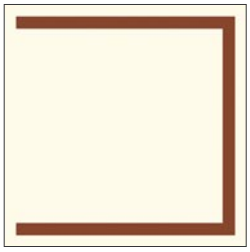

Figure 2. *result.ppm*

De tels moteurs de rendu au format pixel sous AGG réalisent des opérations fondamentales de mélange alpha dans la mémoire image résultante. Dans la mesure où ils ne réalisent aucun découpage, leur utilisation directe n'est pas très fiable. Ainsi, si les coordonnées dépassent la plage correspondante, vous obtiendrez un comportement indéfini, et la plupart du temps des fautes de segmentation. Ce qui ne veut pas dire que la conception est mauvaise, mais que vous pouvez rédiger votre propre espace couleur ainsi que vos propres moteurs de rendu au format pixel, sans vous soucier du découpage, fourni à un niveau supérieur. Il est ainsi possible de rédiger un moteur de rendu capable de fonctionner dans un autre espace de couleur, comme par exemple XYZ ou Lab, et de définir votre propre espace de couleur. Ces ajouts n'affectent en rien les autres composants d'AGG. A l'heure actuelle, AGG propose des formats de pixels RGB et RGBA à 15, 16, 24, et 32 bits.

L'adaptateur de masque alpha permet d'utiliser un canal de transparence supplémentaire au moment du rendu. Théoriquement, il peut s'agir de n'importe quel type d'adaptateur. Le masque alpha n'est qu'un exemple parmi tant d'autres.

Le moteur de rendu élémentaire (renderer_base) accepte un moteur de rendu au format pixel en tant qu'argument de modèle. Son premier objectif consiste à découper au niveau des lignes. L'interface de ce moteur est essentiellement identique à celle des moteurs de rendu au format pixel. Vous trouverez également renderer_mclip, dont l'utilisation reste identique en plus de sa fonctionnalité de découpage en un certain nombre de rectangles arbitraires.

Un des concepts clé d'AGG est le *scanline*, conteneur composé d'un cer-

tain nombre d'intervalles horizontaux pouvant transporter des informations sur l'anticrénelage. Le moteur de rendu de perspectives décompose ces dernières en un nombre d'intervalles. Dans les cas les plus simples (comme le remplissage solide), il fait appel à un moteur de rendu classique. Dans les cas plus complexes, il fait appel à un générateur d'intervalles. Une fois un intervalle de couleur obtenu, il le mélange à la toile.

Le générateur d'intervalles est utilisé dans tous les cas généralement plus complexes qu'un simple remplissage solide. Il s'agit d'un mécanisme général capable de produire toute sorte d'intervalles de couleur (dégradés, ombrage Gouraud, transformations d'images, remplissage de modèles, et ainsi de suite). Ce mécanisme permet notamment de transformer une partie d'image liée à une forme arbitraire au contour anticrénelé. Le générateur d'intervalles peut se composer de l'ensemble de l'architecture pipeline. Par exemple, il peut consister en un transformateur d'images combiné à un dégradé alpha (transparence).

Listing 1. *Fichier "example_pipeline1.cpp"*

```
#include <stdio.h>
#include <string.h>
#include "agg_pixfmt_rgb24.h"
#include "agg_renderer_base.h"
#include "agg_renderer_scanline.h"
#include "agg_scanline_u.h"
#include "agg_rasterizer_scanline_aa.h"
#include "agg_path_storage.h"
enum {
    frame_width = 200,
    frame_height = 200
};
bool write_ppm(const unsigned char* buf, unsigned width, unsigned height,
               const char* file_name) {
    FILE* fd = fopen(file_name, "wb");
    if(fd) {
        fprintf(fd, "P6 %d %d 255 ", width, height);
        fwrite(buf, 1, width * height * 3, fd);
        fclose(fd);
        return true;
    }
    return false;
}
int main() {
    unsigned char* buffer = new unsigned char[frame_width * frame_height * 3];
    agg::rendering_buffer rbuf(buffer, frame_width, frame_height,
        frame_width * 3);
    agg::pixfmt_rgb24 pixf(rbuf);
    agg::renderer_base<agg::pixfmt_rgb24> ren_base(pixf);
    ren_base.clear(agg::rgba8(255, 250, 230));
    agg::scanline_u8 sl;
    agg::rasterizer_scanline_aa<> ras;
    agg::renderer_scanline_aa_solid<
        agg::renderer_base<agg::pixfmt_rgb24> > ren_sl(ren_base);
    agg::path_storage path;
    path.remove_all(); // Pas obligatoire dans ce cas
    path.move_to(10, 10);
    path.line_to(frame_width-10, 10);
    path.line_to(frame_width-10, frame_height-10);
    path.line_to(10, frame_height-10);
    path.line_to(10, frame_height-20);
    path.curve4(frame_width-20, frame_height-20, frame_width-20, 20, 10, 20);
    ras.add_path(path);
    ren_sl.color(agg::rgba8(120, 60, 0));
    agg::render_scanlines(ras, sl, ren_sl);
    write_ppm(buffer, frame_width, frame_height, "result.ppm");
    delete [] buffer;
    return 0;
}
```

Figure 3. *Les résultats obtenus après avoir ajouté /agg2/src/agg_curves.cpp à la liste de compilation*

Le convertisseur de perspectives accepte un certain nombre de polygones arbitraires en tant que données d'entrée pour produire des perspectives anticrénelées. Il s'agit de la première technique de rendu d'AGG. Autrement dit, la seule forme susceptible d'être rendue est le polygone (poly-polygone

Listing 2a. *GNU C++*

```
g++ -I/agg2/include example_
                        pipeline1.cpp
      /agg2/src/agg_rasterizer_
                        scanline_
                        aa.cpp
      /agg2/src/agg_path_storage.cpp
      /agg2/src/agg_bezier_arc.cpp
      /agg2/src/agg_trans_affine.cpp
```

Listing 2b. *VC++*

```
cl -I/agg2/include example_
                        pipeline1.cpp
      /agg2/src/agg_rasterizer_
                        scanline_
                        aa.cpp
      /agg2/src/agg_path_storage.cpp
      /agg2/src/agg_bezier_arc.cpp
      /agg2/src/agg_trans_affine.cpp
```

Listing 3. *Pipelline vectorielle*

```
agg::conv_curve <agg:path_storage>
    curve(path);
agg::conv_stroke <agg::conv_curve<
    agg::path_storage > stroke(
    curve);
stroke.width(6.0);
ras.add_path(curve);
ren_sl.color(agg::rgba8 (160, 180,
    80));
agg::render_scanlines (ras, sl,
    ren_sl);
ras.add_path(stroke);
ren_sl.color(agg::rgba8 (120, 100,
    0));
agg::render_scanlines (ras, sl,
    ren_sl);
```

Listing 4. *Deux pipelines*

```
#include <stdio.h>
#include <string.h>
#include "agg_pixfmt_rgb24.h"
#include "agg_renderer_base.h"
#include "agg_renderer_scanline.h"
#include "agg_scanline_u.h"
#include "agg_rasterizer_scanline_aa.h"
#include "agg_path_storage.h"
#include "agg_conv_curve.h"
#include "agg_conv_stroke.h"
#include "agg_conv_transform.h"
#include "agg_trans_affine.h"
enum {
    frame_width = 200,
    frame_height = 200 };
bool write_ppm(const unsigned char* buf, unsigned width, unsigned height,
               const char* file_name) {
    FILE* fd = fopen(file_name, "wb");
    if(fd) {
        fprintf(fd, "P6 %d %d 255 ", width, height);
        fwrite(buf, 1, width * height * 3, fd);
        fclose(fd);
        return true; }
    return false; }
int main() {
    unsigned char* buffer = new unsigned char[frame_width * frame_height * 3];
    agg::rendering_buffer rbuf(buffer,frame_width,frame_height,frame_width * 3);
    agg::pixfmt_rgb24 pixf(rbuf);
    agg::renderer_base<agg::pixfmt_rgb24> ren_base(pixf);
    ren_base.clear(agg::rgba8(255, 250, 230));
    agg::scanline_u8 sl;
    agg::rasterizer_scanline_aa<> ras;
    agg::renderer_scanline_aa_solid<
        agg::renderer_base<agg::pixfmt_rgb24> > ren_sl(ren_base);
    agg::path_storage path;
    path.remove_all(); // Pas obligatoire dans ce cas
    path.move_to(10, 10);
    path.line_to(frame_width-10, 10);
    path.line_to(frame_width-10, frame_height-10);
    path.line_to(10, frame_height-10);
    path.line_to(10, frame_height-20);
    path.curve4(frame_width-20, frame_height-20, frame_width-20, 20, 10, 20);
    path.close_polygon();
    agg::trans_affine matrix;
    matrix *= agg::trans_affine_translation(-frame_width/2, -frame_height/2);
    matrix *= agg::trans_affine_rotation(agg::deg2rad(35.0));
    matrix *= agg::trans_affine_scaling(0.4, 0.75);
    matrix *= agg::trans_affine_translation(frame_width/2, frame_height/2);
    agg::conv_transform<agg::path_storage, agg::trans_affine> trans(path,matrix);
    agg::conv_curve<
        agg::conv_transform<
            agg::path_storage, agg::trans_affine> > curve(trans);
    agg::conv_stroke<
        agg::conv_curve<
            agg::conv_transform<agg::path_storage,
                                agg::trans_affine> > > stroke(curve);
    stroke.width(6.0);
    ras.add_path(curve);
    ren_sl.color(agg::rgba8(160, 180, 80));
    agg::render_scanlines(ras, sl, ren_sl);
    ras.add_path(stroke);
    ren_sl.color(agg::rgba8(120, 100, 0));
    agg::render_scanlines(ras, sl, ren_sl);
    write_ppm(buffer, frame_width, frame_height, "result.ppm");
    delete [] buffer;
    return 0; }
```

pour être exact). Si vous devez dessiner une ligne, il vous faut calculer au moins quatre points censés définir son contour. Ceci peut vous sembler à première vue fastidieux, mais il n'en est rien. En réalité, l'algorithme utilise l'exactitude des sous-pixels pour rendre correctement n'importe quelle forme, même lorsqu'un seul pixel est traversé par plusieurs arêtes. Vous obtiendrez ainsi un résultat toujours cohérent, quelle que soit sa dimension. L'algorithme permet de garantir l'absence de défaut.

Ce concept a été emprunté au convertisseur du moteur de polices FreeType de David Turner (*http://www.freetype.org*). David m'a gentiment permis de réécrire le convertisseur pour C++ et de le sortir de manière indépendante.

Une telle conception (convertisseur -> moteur scanline_renderer -> moteur base_renderer -> format pixel) permet d'implémenter des algorithmes intéressants, comme des moteurs de rendu optimisés pour les triplets de couleur, tel que ClearType de Microsoft, par exemple, et peut s'appliquer à toutes sortes de primitives (pas seulement du texte).

Les moteurs de rendu de contours relèvent encore d'un autre algorithme permettant de dessiner des lignes anticrénelées. L'utilisation de cet algorithme est plus limitée que celle du convertisseur de perspectives, mais présente également de nombreux avantages :

- Fonctionne 2 à 3 fois plus vitre que le convertisseur de perspectives.

Figure 4. *Les résultats obtenus après avoir ajouté /agg2/src/agg_vcgen_stroke.cpp à la liste de compilation*

Figure 5. *Résultats obtenu*

- Vous permet de disposer de la zone anticrénelée de n'importe quel profil et largeur.
- L'algorithme d'anticrénelage repose sur la distance, contrairement au convertisseur de perspectives. Dans la plupart des cas, il vous permet de produire un meilleur résultat visuel.
- Et avantage le plus important, l'algorithme vous permet d'utiliser une image arbitraire en tant que modèle de lignes. Il s'agit d'une fonctionnalité fondamentale au moment de rendre des cartes géographiques haut de gamme. En règle générale, ce type d'algorithmes est réservé au dessin rapide de lignes polygonales relativement fines.

Le moteur de rendu de contours n'est qu'un adaptateur chargé d'unifier l'utilisation du convertisseur de contours solides et du convertisseur des modèles d'images.

Enfin, l'architecture pipeline vertex se compose d'un certain nombre de convertisseurs. Chacun d'eux accepte une source vertex abstraite en tant que donnée d'entrée et fonctionne comme une autre source de vertex (expose l'interface de la source vertex), de manière à vous permettre de construire des architectures pipelines plus complexes. En règle générale, l'architecture pipeline des bibliothèques graphiques est codée en dur, généralement composée d'un décomposeur de courbes (chargé de convertir les courbes en un nombre de segments de lignes courtes), d'un transformateur affine, d'un générateur de lignes en pointillé, d'un générateur de traits, et d'un découpeur de polygones. Sous AGG, vous pouvez avoir autant d'architectures pipeline personnalisées que vous souhaitez. Dans la mesure où les architectures pipelines peuvent prendre des branches, il est possible d'extraire des vertices de n'importe quel point de l'architecture pipeline. Vous pouvez ainsi combiner les convertisseurs selon vos désirs pour obtenir des résultats assez différents. Tous les convertisseurs sont des modèles de classes, permettant de créer des pipelines de manière statique au moment de la compilation, dans un souci d'efficacité, afin d'éviter les appels virtuels. Rien ne vous empêche toutefois de rédiger des classes enveloppantes polymorphiques

Listing 5. *Architecture pipeline*

```
#include <stdio.h>
#include <string.h>
#include "agg_pixfmt_rgb24.h"
#include "agg_renderer_base.h"
#include "agg_renderer_scanline.h"
#include "agg_scanline_u.h"
#include "agg_rasterizer_scanline_aa.h"
#include "agg_path_storage.h"
#include "agg_conv_curve.h"
#include "agg_conv_stroke.h"
#include "agg_conv_transform.h"
#include "agg_trans_affine.h"
#include "agg_conv_segmentator.h"
#include "agg_trans_warp_magnifier.h"
enum {
    frame_width = 200,
    frame_height = 200
};
bool write_ppm(const unsigned char* buf, unsigned width,
               unsigned height, const char* file_name) {
    FILE* fd = fopen(file_name, "wb");
    if(fd) {
        fprintf(fd, "P6 %d %d 255 ", width, height);
        fwrite(buf, 1, width * height * 3, fd);
        fclose(fd);
        return true;
    }
    return false;
}
int main() {
    // Allouer la mémoire image (dans notre cas "manuellement")
    // et créer l'objet image rendu
    unsigned char* buffer = new unsigned char[frame_width * frame_height * 3];
    agg::rendering_buffer rbuf(buffer, frame_width, frame_height, frame_width *
                       3);

    // Créer le format pixel et les moteurs de rendu basiques
    //--------------------
    agg::pixfmt_rgb24 pixf(rbuf);
    agg::renderer_base<agg::pixfmt_rgb24> ren_base(pixf);

    // Pour finir, tâche très simple, comme le nettoyage
    //--------------------
    ren_base.clear(agg::rgba8(255, 250, 230));

    // Créer le conteneur Scanline, le convertisseur Scanline,
    // et le moteur de rendu Scanline pour le remplissage solide.
    //--------------------
    agg::scanline_u8 sl;
    agg::rasterizer_scanline_aa<> ras;
    agg::renderer_scanline_aa_solid<
        agg::renderer_base<agg::pixfmt_rgb24> > ren_sl(ren_base);

    // Créer l'objet source vertex (chemin), dans notre cas, il
    //  s'agit de path_storage puis former le chemin.
    //--------------------
    agg::path_storage path;
    // Pas obligatoire dans ce cas
    path.remove_all();
    path.move_to(10, 10);
    path.line_to(frame_width-10, 10);
    path.line_to(frame_width-10, frame_height-10);
    path.line_to(10, frame_height-10);
    path.line_to(10, frame_height-20);
    path.curve4(frame_width-20, frame_height-20, frame_width-20, 20, 10, 20);
    path.close_polygon();
```

Listing 5. *Architecture pipeline*

```
    // pipeline vectorielle
    //-----------------------
    agg::trans_affine matrix;

    matrix *= agg::trans_affine_translation(-frame_width/2, -frame_height/2);
    matrix *= agg::trans_affine_rotation(agg::deg2rad(35.0));
    matrix *= agg::trans_affine_scaling(0.3, 0.45);
    matrix *= agg::trans_affine_translation(frame_width/2, frame_height/2);

    agg::trans_warp_magnifier lens;
    lens.center(120, 100);
    lens.magnification(3.0);
    lens.radius(18);

    agg::conv_curve<agg::path_storage> curve(path);
    agg::conv_segmentator<agg::conv_curve<agg::path_storage> > segm(curve);
    agg::conv_transform<
        agg::conv_segmentator<
            agg::conv_curve<
                agg::path_storage> >,
        agg::trans_affine> trans_curve(segm, matrix);
    agg::conv_transform<
        agg::conv_transform<
            agg::conv_segmentator<
                agg::conv_curve<
                    agg::path_storage> >,
            agg::trans_affine>,
        agg::trans_warp_magnifier> trans_warp(trans_curve, lens);
    agg::conv_stroke<
        agg::conv_segmentator<
            agg::conv_curve<
                agg::path_storage> > > stroke(segm);
    agg::conv_transform<
        agg::conv_stroke<
            agg::conv_segmentator<
                agg::conv_curve<
                    agg::path_storage> > >,
        agg::trans_affine> trans_stroke(stroke, matrix);

    agg::conv_transform<
        agg::conv_transform<
            agg::conv_stroke<
                agg::conv_segmentator<
                    agg::conv_curve<
                        agg::path_storage> > >,
            agg::trans_affine>,
        agg::trans_warp_magnifier> trans_warp_stroke(trans_stroke, lens);

    stroke.width(6.0);
    ras.add_path(trans_warp);
    ren_sl.color(agg::rgba8(160, 180, 80));
    agg::render_scanlines(ras, sl, ren_sl);

    ras.add_path(trans_warp_stroke);

    ren_sl.color(agg::rgba8(120, 100, 0));
    agg::render_scanlines(ras, sl, ren_sl);
    //-----------------------

    // Rédiger la mémoire tampon dans le fichier result.ppm et
    //  libérer la mémoire.
    //-----------------------
    write_ppm(buffer, frame_width, frame_height, "result.ppm");
    delete [] buffer;
    return 0;
}
```

très simples lors de l'exécution. Le seul coût supplémentaire consistera en un appel virtuel par vertex et par élément de la pipeline (convertisseur).

Exemple

Afin d'illustrer le fonctionnement d'AGG, nous allons présenter un exemple d'application dont l'objectif est de créer des types basiques pour le rendu. Par souci de clarté et de simplicité, le présent exemple est une application de console chargée de produire des fichiers de rendu (result.ppm) au format simple. Vous pouvez afficher ces fichiers au moyens de nombreux afficheurs, comme par exemple IrfanView (*http://www.irfanview.com/*).

Par exemple, entrez dans le Listing 1 et nommer le fichier "example_pipeline1.cpp". Compilez le et reliez le au Listing 2 (a) grâce à AGG (en utilisant GNU C++) ou au Listing 2 (b) (pour VC++). Il vous faudra remplacer "/agg2/" par le chemin en question vers la bibliothèque. Nous avons exposé dans la Figure 2 ce que vous êtes censé obtenir avec result.ppm.

Vous disposez pour le moment d'une pipeline nulle (vide), dans laquelle tous les points sont interprétés sous forme de commandes *move-to/line-to*. C'est la raison pour laquelle l'appel de path.curve4 n'a aucun effet.

Afin d'ajouter un convertisseur de courbes (ou, en d'autres termes, un aplatisseur de courbes) chargé de décomposer les courbes Bézier en un nombre de segments de lignes courts (example_pipeline2.cpp), il suffit d'ajouter la ligne suivante :

```
#include "agg_conv_curve.h
```

puis de procéder aux modifications suivantes dans le Listing 1 :

```
// Pipeline vectorielle
//-----------------------
agg::conv_curve <agg::path_
storage>curve(path);
ras.add_path(curve);
//-----------------------
```

Nous avons exposé dans la Figure 3 les résultats obtenus après avoir ajouté /agg2/src/agg_curves.cpp à la liste de compilation.

Afin de dessiner un trait, il suffit d'ajouter la ligne suivante :

```
#include "agg_conv_stroke.h
```

puis de procéder aux modifications suivantes dans le Listing 1 :

```
// pipeline vectorielle
//-----------------------
agg::conv_curve<agg::path_storage>
                  curve(path);
agg::conv_stroke<agg::conv_curve<agg:
                  :path_storage >>
                  stroke(curve);
stroke.width(6.0);
ras.add_path(stroke);
//-----------------------
```

Nous avons exposé dans la Figure 4 les résultats obtenus après avoir ajouté /agg2/src/agg_vcgen_stroke.cpp à la liste de compilation.

Bien sûr, vous pouvez également régler la largeur de la ligne, son débit, et sa jonction.

Le chemin peut comprendre autant de polygones fermés et ouverts séparés par la commande move_to. Afin de fermer un polygone, il suffit d'appeler :

```
path.close_polygon();
```

Afin de dessiner un polygone rempli de traits, il vous suffit de changer le Listing 1 par le Listing 3. L'architecture pipeline se compose désormais de deux convertisseurs consécutifs (courbe et trait) utilisables de la même manière.

L'étape suivante consiste à réaliser les transformations affines, puis il faut décider où les ajouter. La réponse bien évidemment "dépendra" d'un certain nombre de paramètres. Vous pouvez ajouter le transformateur affine avant le convertisseur de courbes, de manière à ce qu'il ne traite que quelques points seulement. Attention toutefois, le convertisseur de traits générera un trait comme s'il s'agissait de la forme originale. Vous remarquerez dans le Listing 4 deux pipelines (chemin -> moteur conv_curve -> moteur conv_transform, et chemin -> moteur conv_curve -> moteur conv_stroke -> moteur conv_transform). A première vue, il semble que le trait soit plus fin, mais le procédé est bien plus complexe. Vous aurez remarqué que la largeur du trait n'est pas uniforme. Nous avons volontairement réglé différents coefficients de dimensions grâce à X et Y, afin de vous démontrer la possibilité de contrôler les résultats obtenus en modifiant l'ordre des convertisseurs.

Après le moteur conv_curve, l'architecture pipeline se diversifie. Pour plus d'efficacité, il serait plus judicieux de conserver la pipeline de remplissage telle qu'exposée dans le Listing 1 (chemin -> moteur conv_transform -> moteur conv_curve). Le Listing 4 permet de démontrer la possibilité d'élaborer des pipelines beaucoup plus complexes.

Nous allons désormais vous présenter les effets d'une transformation non-linéaire : agrandissement de distorsions circulaires. Dans la mesure où les transformations ne sont pas linéaires, il est impossible de transformer des vertices tels quels. Il faut tout d'abord préparer le chemin de sorte que les vecteurs initiaux soient décomposés en nombreux petits segments linéaires. Bien évidemment, vous pouvez procéder ainsi au moment d'ajouter des vertices au stockage du chemin, mais une meilleure solution est également possible. Vous pouvez utilisez un convertisseur intermédiaire supplémentaire (conv_segmentator) chargé de segmenter les vecteurs longs. Il vous faudra également inclure deux fichiers :

```
#include "agg_conv_segmentator.h?
#include "agg_trans_warp_magnifier.h"
```

Listing 6. *Ligne de commande pour la compilation*

```
g++ -I/agg2/include example_pipeline7.cpp
    /agg2/src/agg_rasterizer_scanline_aa.cpp
    /agg2/src/agg_path_storage.cpp
    /agg2/src/agg_bezier_arc.cpp
    /agg2/src/agg_trans_affine.cpp
    /agg2/src/agg_curves.cpp
    /agg2/src/agg_vcgen_stroke.cpp
    /agg2/src/agg_vpgen_segmentator.cpp
    /agg2/src/agg_trans_warp_magnifier.cpp
```

Et vous obtiendrez une architecture pipeline identique à celle exposée dans le Listing 5. Nous avons exposé dans le Listing 6 la ligne de commande pour la compilation, et dans la Figure 5 le résultat obtenu.

Ces exemples illustrent à merveille le principe majeur régissant la conception avec AGG : avoir le contrôle total de votre modèle de rendu et des capacités de l'outil. Telles quelles, les déclarations peuvent vous sembler complexes, mais il est possible de les simplifier grâce à l'utilisation des définitions de types. Vous pouvez également créer une seule classe enveloppante puis l'utiliser sous forme de bibliothèque graphique conventionnelle. Le dernier exemple illustre la simplicité avec laquelle il est possible d'étendre cette fonctionnalité. Dans ces exemples, l'architecture pipeline est définie de manière statique au moment de la compilation, procédé qui conviendra à la plupart des cas.

Les pipelines de représentation sont organisées sur le même modèle, mais sont généralement plus directes. Une fois la forme vectorielle tramée, vous pouvez :

- Remplir la forme d'une couleur dense avec d'éventuels effets de transparence.
- Remplir la forme avec un dégradé arbitraire. Toutes les transformations possibles sont également applicables aux dégradés.
- Remplir la forme d'une image transformée. De nombreux filtres anticrénelage sont disponibles, du filtre simple bilinéaire, aux filtres haut de gamme de Lancosz et Blackman.
- Remplir la forme d'un modèle arbitraire (avec ou sans transformations).
- Appliquer un masque alpha.
- Appliquer un certain nombre d'opérations algébriques booléennes pour les perspectives, comme l'intersection, l'union, la différence, et l'opérateur XOR entre deux ou plusieurs formes mises en perspective.

AGG comporte bien d'autres algorithmes dignes d'intérêt, comme par exemple les images tramées en tant que modèles de lignes. Il s'agit d'un mécanisme puissant pour la cartographie et les applications similaires, jusqu'ici inégalé.

Vous pouvez consulter de nombreux autres exemples, dont des applications interactives multi-plate-formes sur le site Web *http://antigrain.com*. ■