Laboratoire informatique – SIMGraphicInterpolation : L'application des interfaces

Table des matières

Description du programme

Le laboratoire **SIMGraphicInterpolation** est une application permettant d'effectuer des dessins de lignes et de triangles sur un écran de vue (*viewport*) par la technique de l'interpolation linéaire. Le tout sera visualisé sur un fichier image de format png qui sera lu par un logiciel d'imagerie (ex : *Visionneuse de photos Windows*). Les couleurs des dessins réalisés seront à couleur unique, à couleur interpolée ou seront déterminées par l'application d'une texture de couleur.

Objectif du laboratoire

Ce laboratoire consistera à se familiariser avec le concept d'interface. L'usage d'interface permettra à différents objets de type **SVector** d'implémenter des fonctionnalités mathématiques communes afin de centraliser les implémentations mathématiques dans une seule classe au lieu de dupliquer l'écriture du code « semblable » dans toutes les classes ayant un comportement vectoriel.

Documentation

Une documentation en lien avec les concepts de ce laboratoire est disponible aux liens suivants :

http://profs.cmaisonneuve.qc.ca/svezina/nyc/note_nyc/NYC_XXI_Chap%206.1.pdf

http://profs.cmaisonneuve.qc.ca/svezina/nyc/note_nyc/NYC_XXI_Chap%206.3.pdf

http://profs.cmaisonneuve.qc.ca/svezina/nyc/note_nyc/NYC_XXI_Chap%206.7.pdf

Programme

Le programme **SIMGraphicInterpolation** est disponible dans un format **jar** au lien suivant :

http://profs.cmaisonneuve.qc.ca/svezina/projet/ray_tracer/download/SIMGraphicInterpolation.jar

Partie 1 : Description de la méthode main

Vous allez commencer par ouvrir le fichier **SIMGraphicInterpolation.java** et lire le contenu de la méthode main() de cette classe. On y retrouve la construction d'un **viewport** où des dessins seront réalisés. L'affichage du viewport sera réalisé par l'appel de la méthode

```
public void writeImage() throws IOException
```

à l'objet viewport. Lancez l'exécution du programme et visualisez le fichier image généré portant le nom de « interpolation.png ». L'image est présentement strictement noire puisqu'aucune écriture n'a été effectuée sur le viewport.

Partie 2 : Dessiner une ligne par interpolation

1- Pour dessiner une ligne dans le **viewport**, vous allez utiliser dans votre méthode main() la méthode

```
private static void displayLine(SViewport viewport, SColor color, SVectorPixel p0, SVectorPixel p1, int iteration)
```

où les paramètres sont définis dans la **javadoc**. Choisissez deux pixels de votre choix dans le **viewport** qui sont situés à l'intérieur des limites de celui-ci afin d'éviter une exception de type **SCoordinateOutOfBoundException** pour définir votre ligne. Choisissez également une couleur de votre choix à l'aide de la construction d'un objet de type **SColor**. Une exception de type **SConstructorException** ou **SReadingException** sera lancée si votre couleur est mal définie.

2- Avant de visualiser la ligne, il faudra effectuer le calcul de l'interpolation linéaire et l'utiliser pour dessiner la ligne. En premier temps, ouvrez le fichier **SVectorUtil.java** et implémentez la méthode

```
public static SVector linearInterpolation(SVector v0, SVector v1, double t) throws SRuntimeException
```

pour réaliser le calcul de l'interpolation linéaire. Inspirez-vous de la **javadoc** pour réaliser votre implémentation.

3- Ouvrez maintenant le fichier **SVectorPixel.java**. Présentement cette classe implémente (**implements**) l'interface **SWriteable** permettant à cette classe d'être écrit sous la forme d'une String dans un fichier texte grâce à la méthode

```
public void write(BufferedWriter bw) throws IOException
```

De plus, remarquez que cette classe permet l'opération de l'addition et de la multiplication par un scalaire. Cependant, un **SVectorPixel** n'est pas un **SVector** car elle n'implémente pas cette interface. Pour remédier à cette situation, ajoutez l'implémentation de l'interface **SVector** à cette classe

```
public class SVectorPixel implements SWriteable, SVector {    …    }
```

et implémentez la méthode

```
public SVector add(SVector v);
```

afin qu'un **SVectorPixel** puisse implémenter adéquatement l'interface **SVector**. Par la suite, vous pourrez utiliser un **SVectorPixel** dans la méthode d'interpolation linéaire de la classe **SVectorUtil**, car il est maintenant considéré comme étant un **SVector**.

4- Retournez au fichier **SIMGraphicInterpolation.java** et implémentez la méthode

```
private static void displayLine(SViewport viewport, SColor color, SVectorPixel p0, SVectorPixel p1, int iteration)
```

en utilisant la méthode

```
public static SVector linearInterpolation(SVector v0, SVector v1, double t) throws SRuntimeException
```

de la classe **SVectorUtil**. Vous devrez effectuez un balayage de la variable *t* de 0.0 à 1.0 sur un nombre d'itération égal à la variable itération passée en paramètre pour y calculer, par interpolation linéaire, chacun des pixels que vous allez colorer de la couleur sélectionnée. Pour colorer un pixel du viewport, appelez la méthode

```
public void setColor(SVectorPixel p, SColor color) throws SCoordinateOutOfBoundException
```

à l'objet viewport pour chacun des pixels calculés lors de votre interpolation linéaire.

5- Exécutez votre programme et visualisez votre ligne. Vous pouvez en afficher plus d'une si vous le désirez. Si des exceptions sont lancées durant l'exécution du programme, analyser leur nature et rectifiez vos paramètres d'entrée de vos méthodes.

Partie 2 : Dessiner un triangle par interpolation

1- Pour dessiner un triangle dans le **viewport**, vous allez utiliser dans votre méthode main() la méthode

```
private static void displayTriangle(SViewport viewport, SColor color, SVectorPixel p0, SVectorPixel p1,
SVectorPixel p2, int iteration)
```

où les paramètres sont définis dans la **javadoc**. Choisissez trois pixels dans le **viewport** qui sont situés à l'intérieur des limites de celui-ci pour définir votre triangle et choisissez une couleur.

2- Avant de visualiser le triangle, il faudra effectuer le calcul de l'interpolation en coordonnée barycentrique et l'utiliser pour dessiner le triangle. En premier temps, ouvrez le fichier **SVectorUtil.java** et implémentez la méthode

```
public static SVector linearBarycentricInterpolation(SVector v0, SVector v1, SVector v2, double t1,
double t2)   throws SRuntimeException
```

pour réaliser le calcul de l'interpolation. Inspirez-vous de la **javadoc** pour réaliser votre implémentation.

3- Retournez au fichier **SIMGraphicInterpolation.java** et implémentez la méthode

```
private static void displayTriangle(SViewport viewport, SColor color, SVectorPixel p0, SVectorPixel p1,
SVectorPixel p2, int iteration)
```

en utilisant la méthode

```
public static SVector linearBarycentricInterpolation(SVector v0, SVector v1, SVector v2, double t1,
double t2)   throws SRuntimeException
```

de la classe **SVectorUtil**. Effectuez un balayage de la variable t_1 et t_2 de 0.0 à 1.0 en respectant la contrainte $t_1 + t_2 < 1$ sur un nombre d'itération égal à la variable itération passée en paramètre. N'oubliez pas de colorier vos pixels dans le viewport.

4- Exécutez votre programme et visualisez votre triangle. Vous pouvez afficher plusieurs triangles si vous le désirez. Si des exceptions sont lancées durant l'exécution du programme, analyser leur nature et rectifiez vos paramètres d'entrée de vos méthodes.

Partie 3 : Dessiner une ligne par interpolation avec interpolation des couleurs

1- Pour dessiner une ligne dans le **viewport** dont la couleur va changer graduellement d'une couleur de référence à une autre par interpolation linéaire, vous allez utiliser dans votre méthode main() la méthode

```
private static void displayLineInterpolatedColor(SViewport viewport, SColor color0, SColor color1,
SVectorPixel p0, SVectorPixel p1, int iteration)
```

où les paramètres sont définis dans la **javadoc**. Choisissez deux pixels dans le **viewport** qui sont situés à l'intérieur des limites pour définir votre ligne et choisissez deux couleurs différentes que vous allez associer à vos deux points de votre ligne.

2- Avant de visualiser la ligne, il faudra permettre à une couleur d'être interpolée par la méthode de la classe **SVectorUtil**. Pour ce faire, vous allez modifier le fichier **SColor.java**. Vous allez permettre à la classe **SColor** d'implémenter l'interface **SVector** et vous allez implémenter les méthodes nécessaires.

3- Retournez au fichier **SIMGraphicInterpolation.java** et implémentez la méthode

```
private static void displayLineInterpolatedColor(SViewport viewport, SColor color0, SColor color1,
SVectorPixel p0, SVectorPixel p1, int iteration)
```

en utilisant la méthode

```
public static SVector linearInterpolation(SVector v0, SVector v1, double t) throws SRuntimeException
```

de la classe **SVectorUtil**. Utilisez une approche similaire à ce que vous avez déjà réalisée, mais ajoutez simplement un calcul d'interpolation de la couleur pour afficher dans le **viewport** la couleur interpolée.

4- Exécutez votre programme et visualisez votre ligne à couleur interpolée. Vous pouvez afficher plusieurs lignes si vous le désirez. Si des exceptions sont lancées durant l'exécution du programme, analyser leur nature et rectifiez vos paramètres d'entrée de vos méthodes.

Partie 4 : Dessiner un triangle par interpolation avec interpolation des couleurs

1- Pour dessiner un triangle dans le **viewport** dont la couleur va changer graduellement d'une couleur de référence à une autre par interpolation en coordonnée barycentrique, vous allez utiliser dans votre méthode main() la méthode

```
private static void displayTriangleInterpolatedColor(SViewport viewport, SColor color0, SColor color1,
SColor color2, SVectorPixel p0, SVectorPixel p1, SVectorPixel p2, int iteration)
```

où les paramètres sont définis dans la **javadoc**. Avec cette méthode, votre triangle aura trois couleurs de référence situées aux trois extrémités du triangle. Choisissez trois pixels dans le **viewport** qui sont situés à l'intérieur des limites pour définir votre triangle et choisissez trois couleurs différentes de votre choix.

2- Dans le fichier **SIMGraphicInterpolation.java**, implémentez la méthode

```
private static void displayTriangleInterpolatedColor(SViewport viewport, SColor color0, SColor color1,
SColor color2, SVectorPixel p0, SVectorPixel p1, SVectorPixel p2, int iteration)
```

en utilisant la méthode

```
public static SVector linearBarycentricInterpolation(SVector v0, SVector v1, SVector v2,
double t1, double t2)   throws SRuntimeException
```

de la classe **SVectorUtil**. Utilisez une approche similaire à ce que vous avez déjà réalisée, mais ajoutez simplement un calcul d'interpolation des couleurs pour afficher dans le **viewport** la couleur interpolée.

4- Exécutez votre programme et visualisez votre triangle à couleur interpolée. Vous pouvez afficher plusieurs triangles si vous le désirez. Si des exceptions sont lancées, analyser leur nature et rectifiez vos paramètres d'entrée de vos méthodes.

Partie 5 : Dessiner un triangle par interpolation avec application d'une texture

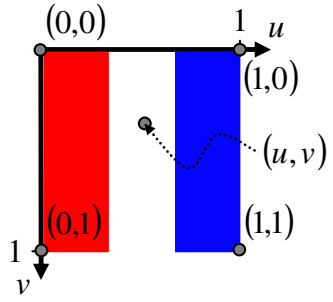
1- Pour dessiner un triangle dans le **viewport** dont les couleurs vont être déterminées par une texture, vous allez utiliser dans votre méthode main() la méthode

```
private static void displayTriangleInterpolatedTexture(SViewport viewport, STexture texture, SVectorUV uv0,
SVectorUV uv1, SVectorUV uv2, SVectorPixel p0, SVectorPixel p1, SVectorPixel p2, int iteration)
```

où les paramètres sont définis dans la **javadoc**. Avec cette méthode, votre triangle aura trois coordonnées de texture *uv* correspondant aux trois extrémités du triangle. Ces coordonnées auront une référence dans une texture de couleur correspondant à un fichier image externe au programme (ex : canada.png).

Les coordonnées de texture correspondant aux quatre coins d'une image sont les suivantes :

- coin supérieur gauche : (0.0, 0.0)
- coin supérieur droit : (1.0, 0.0)
- coin inférieur gauche : (0.0, 1.0)
- coint inférieur droit : (1.0, 1.0)



Texture de couleur en coordonnée *uv* correspond au drapeau de la France.

Construisez une texture de type **STexture** avec la classe **STextureLoader** en utilisant la méthode

```
public STexture loadTexture(String file_name) throws SLoaderException
```

à l'objet de type **STextureLoader**. Puisque cet objet lance des exceptions de type **SLoaderException**, vous devrez utiliser un bloc « **try-catch** » pour gérer l'exception si elle se présente lors de l'exécution du programme (ex : Le nom du fichier n'est pas trouvé).

2- Dans le fichier **SIMGraphicInterpolation.java**, implémentez la méthode

```
private static void displayTriangleInterpolatedTexture(SViewport viewport, STexture texture, SVectorUV uv0,
SVectorUV uv1, SVectorUV uv2, SVectorPixel p0, SVectorPixel p1, SVectorPixel p2, int iteration)
```

en utilisant la méthode

```
public static SVector linearBarycentricInterpolation(SVector v0, SVector v1, SVector v2,
double t1, double t2)   throws SRuntimeException
```

de la classe **SVectorUtil**. Utilisez une approche similaire à ce que vous avez déjà réalisée, mais ajoutez simplement un calcul d'interpolation des coordonnées de texture *uv* pour afficher dans le **viewport** la couleur de la texture dans votre triangle au bon endroit.

Vous remarquerez que la classe **SVectorUV** n'est pas adéquatement écrite pour être utilisée dans le calcul de l'interpolation linéaire en coordonnée barycentrique. Modifiez la classe **SVectorUV** pour obtenir les fonctionnalités désirées.

3- Exécutez votre programme et visualisez votre triangle avec texture de couleur. Vous pouvez afficher plusieurs triangles si vous le désirez (avec des textures différentes également). Si des exceptions sont lancées, analyser leur nature et rectifiez vos paramètres d'entrée de vos méthodes.