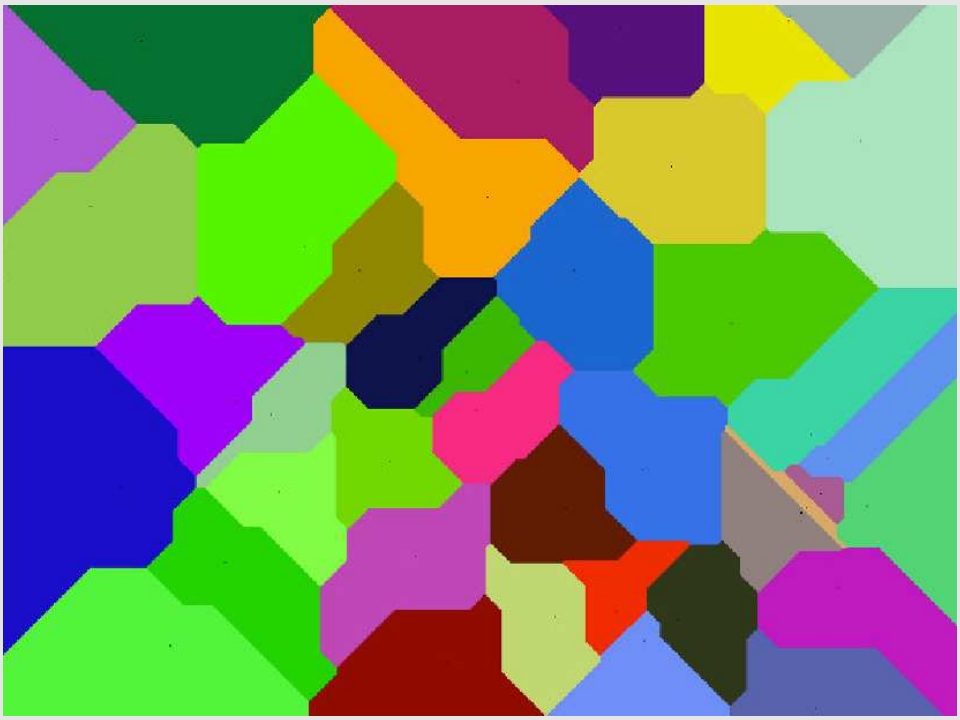
IN 103 - TD 02

Solution

ENSTA ParisTech

On souhaite simuler l'extension d'affreuses créatures rampantes s'étalant progressivement autour de leur point d'apparition...

1. On choisit n points initiaux P (par exemple au hasard) auxquels on attribue une couleur.
2. On attribue aux autres points de l'écran la nouvelle couleur telle que chacun soit de la même couleur que la colonie qui l'a conquis.



Q1 Quelle structure de données sous-jacente peut être utile ? Pourquoi (vague idée de comment l'algorithme va fonctionner) ?

Solution

L'idée est que pour chaque point que l'on colorie, on va mémoriser ses voisins pour les colorier à leur tour avec la même couleur. Il faut donc que les points «étendent leur couleur » à leur voisins «en parallèle ».

Donc, lorsque l'on a colorié un point, il faut se souvenir qu'il faudra colorier ses voisins, mais avant, s'occuper des autres points en suspend. La structure de file est très adaptée. Initialement, on va insérer les points de départ.

Ensuite, chaque point traité à un instant donné va insérer dans la file ses voisins immédiats

en spécifiant qu'à leur moment d'être traité, ils devront acquérir la même couleur que le point actuel.

Comme les points initiaux sont traités les uns après les autres et qu'ils rajoutent leurs voisins dans la file, et que ces voisins sont traités inductivement de la même manière, la file mémorisera des ≪tranches≫ de points de distance identique autour de chaque nouveau point rencontré.

Autrement dit, après avoir inséré tous les points initiaux, l'algorithme va traiter tous les voisins de distance 1 des points initiaux. Ces derniers étant alors mis en file, une fois que l'on en aura terminé avec les voisins de distance 1 pré-cités, on prendra les voisins de distance 1 ... des voisins de distance 1 des points initiaux. Donc de distance 2. Etc.

Q2 Quelle est la forme des données à mémoriser dans cette structure ?

Solution

La structure d'information mémorisée dans la file devra donc contenir les coordonnées du point ainsi que la couleur (composantes R, G, B) de laquelle il devra être colorié.

Q3 Esquissez l'algorithme pour effectuer cette décomposition.

Solution

Principe de l'algorithme : Il faudra bien entendu ne pas re-colorier un point déjà colorié !

```
Mettre dans la file les points initiaux avec leur couleur.
Tant que la file n'est pas vide faire:
  Récupérer le point en tête de la file
  L'afficher selon sa couleur
  Pour chacun des 8 points voisins autour (voisinage 3x3)
    Si point voisin non traité et dans les limites de l'écran
      Insérer le point voisin dans la file en lui attribuant la
      couleur du point courant
```

Dans le fichier `queue.(c|h)` vous sont données les fonctions de manipulation de la structure de donnée discuté précédemment. La suite du TD vise à implémenter l'algorithme de coloriage.

Q4 Quelle est la taille maximale de la structure de mémorisation à créer ?

Solution

Au pire des cas, il peut y avoir une couleur pour chaque point. Donc la taille maximale de la file sera le nombre de points de l'écran : largeur × hauteur.

Q5 Dans quel cas cette structure peut-elle être saturée (par rapport à la taille maximale trouvée dans la question **Q4** ?

Solution

La file peut être pleine dans le cas où l'utilisateur choisit un nombre de points initiaux supérieur au nombre de points possibles à l'écran.

Q6 Comment pouvez-vous mémoriser si un point a déjà été colorié ? Comment allez-vous accéder à cette information ?

Solution

Il suffit de disposer d'un tableau de booléens pouvant contenir l'information pour chaque point de l'écran.

Il y a 2 solutions. Soit on alloue un tableau à 2 dimensions, mais c'est pénible à faire car il faut initialiser chacune des dimensions. Soit on fait un simple tableau unidimensionnel de taille largeur × hauteur et on y accède en indexation linéarisée (c.f. IN102). Autrement dit, l'information sur le point de coordonnées (x, y) se trouve à la case $y \times \text{largeur} + x$.

Bien évidemment, chaque case doit être initialisée à `false` puisque initialement aucun des points n'est colorié.

La couleur d'un point est déterminée par la ≪quantité≫ de rouge, de vert et de bleu qu'elle comporte. Ce sont donc 3 entiers qui sont compris dans $[0; 255]$. Pour choisir une couleur au hasard, il faut donc choisir 3 entiers entre 0 et 255.

Vous avez à disposition la fonction :

`int rand (void)`

qui renvoie un entier pseudo aléatoire entre 0 et une constante prédéfinie (très grande) inclus.

Cette fonction vous servira également pour choisir aléatoirement les coordonnées des points de départ.

Q7 Comment restreindre la valeur retournée par `rand ()` à $[0; 255]$?

Solution

Il suffit de faire un modulo 256 :

rand () % 256 ;

Et pour les coordonnées des points de départ, il suffit de faire un modulo avec la largeur et la hauteur de l'écran.

Pour vous aider dans la suite, vous disposez, dans le fichier `gfxprims.(c|h)` d'une fonction d'affichage d'un point selon ses coordonnées x et y et sa couleur via ses composantes rouge / vert / bleu :

void renderPixel (**int** x, **int** y, Uint8 R, Uint8 G, Uint8 B)

Q8 Écrivez une fonction `fill ()` qui implémente l'algorithme de coloriage. Cette fonction devra prendre en arguments la taille de l'écran (largeur, hauteur) et le nombre de points initiaux à partir desquels calculer le diagramme.

Le fichier `test_fill.c` vous fournit le substrat supplémentaire vous permettant de tester votre calcul de coloriage, en considérant que la fonction de calcul du diagramme a le prototype suivant :

void fill (**int** width, **int** height, **int** nb_init)

où `width` est la largeur de l'écran, `height` sa hauteur et `nb_init` est le nombre de points à choisir au hasard pour déterminer les centres de propagation des couleurs.

Pour compiler votre programme, vous aurez besoin de certaines options de compilation qui suivent :

```
gcc -Wall `sdl-config --cflags --libs` gfxprims.c test_fill.c VOS_FICHIERS
```