Projets : Algorithmes pour la Programmation Graphique (APG)

Merci de lire attentivement et de suivre les recommandations données ci-après pour la validation de l'EC APG. La liste des projets, incluant leurs descriptifs, est données dans la section suivante.

Après avoir choisi vos projets préférés (vous devez en sélectionner 4 différents en les classant par ordre de préférence [1]) en répondant au sondage "Choix du projet" sur le moodle du cours, et obtenu l'accord de l'enseignant qui fournira les affectations définitives le 18 novembre, il sera attendu de votre part :

1. La création, dans les meilleurs délais, sur votre `gitlab` [2] `code.up8.edu` d'un "nouveau projet" correspondant au projet affecté. Le dépôt-projet Git (ou *repository*) doit être accessible en lecture pour l'enseignant (nom d'utilisateur de l'enseignant : `amsi`), ou être public. Vous devez l'utiliser régulièrement pour remonter vos modifications, ces dernières doivent être systématiquement documentées de manière à bien refléter les changements [3]. La régularité et la cohérence de l'évolution de votre dépôt-projet est capitale, elle est significativement [4] prise en compte dans l'évaluation.
2. De travailler sur votre projet : coder. Il n'est pas strictement interdit de glaner des idées çà et là, il est néanmoins **obligatoire** dans ce cas de citer ses sources ; toute omission peut être considérée comme une tentative de plagiat et être lourdement sanctionnée.
3. La création d'un fichier `README` à la racine du dépôt qui : **a)** décrit votre projet (qui, quand, quoi), ses objectifs, ce qui a été réalisé et ce qui reste à faire ; **b)** où et comment le récupérer ; **c)** comment l'installer ; **d)** comment l'utiliser.
4. De préparer une présentation de votre travail. Cette présentation peut

1. Vous n'avez pas la garantie d'obtenir un accord pour votre premier choix, néanmoins, dans tous les cas, un de vos 4 choix vous sera accordé. Dans le cas où un des projets est trop sollicité, le champ "Motivation" servira à départager les multiples candidatures sur le projet. Il vous est donc fortement recommandé de bien renseigner ce champ.

2. Si le projet est réalisé en binôme, seul le code source doit être commun ; un seul dépôt Git devra être utilisé pour stocker le code source et répertorier les differents `commit` des deux contributeurs. Le rapport et/ou la présentation restent des travaux strictement personnels.

3. Ici je fais référence aux messages liés aux `git commit`.

4. Par exemple, un projet qui passe de 5% "fait" à 95% la veille de la date limite aura de fortes chances d'être rejeté ; de même qu'un projet qui change du tout au tout durant sa période de vie.

prendre la forme d'une vidéo [5] synthétique de présentation [6] de votre projet, ou d'un rapport structuré et détaillé de votre travail [7], ou les deux. **Dans le cas d'un binôme**, chaque contributeur doit fournir une présentation personnelle [8] du travail conjoint.

5. De déposer, par personne, et uniquement sur le moodle du cours APG, avant la date limite, le fichier `README` cité plus haut, l'archive [9] du code source de votre projet, et le(s) fichier(s) de présentation cité(s) ci-dessus.

5. Au format mp4.
6. Je vous conseille l'utilisation d'OBS pour réaliser votre capture vidéo.
7. Au format pdf.
8. Mettre clairement et sans équivoque l'accent sur vos contributions personnelles au projet.
9. Au format tgz.

1 Liste des projets APG

Dans les projets cités ci-dessous, pour tout rendu (au sens graphique du terme), l'utilisation de la bibliothèque de rastérisation développée pendant le cours est obligatoire. Toute amélioration apportée aux algorithmes vus en cours, et, nécessaire au projet, sera valorisée et comptera dans le calcul de la note finale. A cet effet, il sera donc impératif d'en parler dans le rapport, et/ou la présentation vidéo, sans oublier de la mesurer en réalisant des *benchmarks*. Enfin, en cas de réel besoin, l'utilisation d'une bibliothèque de calcul de type « moteur strictement physique » (ni moteur de jeu, ni moteur graphique) telle que *Bullet* est permise [10].

1.1 Génération de MNT par triangulation de Delaunay

1.1.1 Descriptif général

Générer, modéliser, calculer les données d'éclairage et visualiser un Modèle Numérique de Terrain (ou DEM – Digital Elevation Map –, ou Height Map en anglais).

1.1.2 Fonctionnalités attendues

a. Générer à l'aide de la triangulation de Delaunay en échantillonnant aléatoirement et uniformément des points 3D pris dans un MNT existant. On utilisera x (abscisse normalisée du pixel extrait du MNT) et z (ordonnée normalisée du pixel extrait du MNT) pour la triangulation dans le plan (x,z) ; le y (valeur normalisée du pixel (x, z) extrait du MNT) sera simplement ajouté à la coordonnée 3D du sommet et ne servira pas à la triangulation.
b. Modéliser le maillage obtenu en calculant les normales aux sommets (moyenne des normales aux faces) ;
c. Visualiser (plusieurs axes de rotation seront possibles) le terrain rendu avec un éclairage Phong.
d. Texturer le terrain avec du bruit de Perlin pour le grain, plus du coloriage lié à l'altitude et/ou à la pente de chaque facette.

10. Une demande préalable est **obligatoire** et doit être effectuée auprès de l'enseignant pour confirmer le droit d'utiliser la bibliothèque choisie.

1.2 Mosaïque de vidéos par cellules de Voronoï et suivi des cellules par optical flow

Réaliser une mosaïque à partir d'une image en échantillant aléatoirement et uniformément des pixels dans l'espace image et en les utilisant comme sites de cellules d'un diagramme de Voronoï. Effectuer un suivi de la mosaïque dans le cadre d'une séquence d'images (donc une vidéo) en essayant de maintenir un suivi de chaque site en utilisant l'Optical Flow disponible dans la bibliothèque OpenCV.

1.2.1 Descriptif général

1.2.2 Fonctionnalités attendues

a. Le choix de l'algorithme de calcul du diagramme de Voronoï est libre, commencer par le faire sur une image fixe ;
b. Utiliser OpenCV pour ouvrir une vidéo et envoyer chaque frame à l'étape précédente, attention à ne pas refaire un nouvel échantillonnage à chaque frame ; à ce stade le résultat obtenu donne l'impression que la vidéo "glisse" derrière la mosaïque (effet observation derrière un vitrail – *showerdoor effect*) ;
c. Utiliser l'Optical Flow d'OpenCV pour "essayer" déplacer les sites du diagramme pour les pister d'une frame à l'autre ; quand un pixel est perdu, ré-échantillonner aléatoirement un autre qui remplacera le site initial.

1.3 Pacman en mode FPS ou en vue isométrique

1.3.1 Descriptif général

Utiliser les éléments vus/faits en cours pour réaliser soit une version FPS (*First Person Shooter*) soit une version en vue isométrique du jeu Pacman. Utiliser simplement des rectangles texturés pour les fantômes et une sphère texturée pour Pacman.

1.3.2 Fonctionnalités attendues

a. Modéliser un labyrinthe 3D en partant du modèle "vue de haut" : monter les murs, ajouter un sol et un plafond, texturer, réduire l'épaisseur des murs ...
b. Gérer le personnage principal et ses collisions ;
c. Gérer les objets récupérables dans la scène, les ennemis, les collisions et leurs conséquences ;

d. En mode FPS, proposer une vue orthogonale miniature de l'ensemble du labyrinthe. En mode isométrique, permettre de légers changements d'angles de vue.
e. Gérer l'affichage du score, la fin de niveau et le début du suivant.

1.4 Bomberman en vue isométrique – possible en binôme

1.4.1 Descriptif général

Reproduire une version simplifiée du jeu Bomberman avec une vue isométrique :
`https://www.youtube.com/watch?v=lJB277D_lpo`.

1.4.2 Fonctionnalités attendues

a. Sur la base d'un générateur de labyrinthes, modéliser aléatoirement chaque nouveau niveau : on pourra utiliser différents types de cubes comme dans le jeu original.
b. Les joueurs seront simplement modélisés par un cône surmonté d'une sphère, et de couleurs différentes ; des sphères noires pour les bombes ; des sphères jaunes dont le rayon décroit jusqu'à disparaître pour les effet d'explosion ...
c. Gérer toutes les collisions et intéractions liées au jeu ;
d. Gérer l'IA des autres joueurs ;
e. Gérer un mode multi-joueurs "humains" sur le même poste ;
f. Gérer un mode multi-joueurs "humains" en réseau ;

1.5 Benchmark BSP/Quadtree

1.5.1 Descriptif général

Immergé dans un univers de grande taille (par exemple un grand labyrinthe, éclairé, texturé, ...) en vue FPS (*First Person Shooter*), proposer une version non optimisée et la mesurer puis proposer une ou plusieurs optimations (BSP, Quadtree, ...) et mesurer leur apport (principalement en terme de frames par seconde).

1.6 Benchmark Octree

1.6.1 Descriptif général

Immergé dans un univers 3D de grande taille (par exemple un espace contenant des sphères à récupérer et d'autres à éviter) dans lequel nous avons la liberté de nous déplacer dans toutes les directions, proposer une version non optimisée et la mesurer puis proposer une ou plusieurs optimations (frustum-culling, Octree, ...) et mesurer leur apport (principalement en terme de frames par seconde).

1.7 Tetris / Block Out

1.7.1 Descriptif général

Jouer à Tetris, visuellement proche de :
`https://www.youtube.com/watch?v=M3YAnPQp68E`
Ou à Block out : `https://www.youtube.com/watch?v=qTkxmE2AAoo`

1.7.2 Fonctionnalités attendues

a. Jouer à un équivalent de Tetris, en vue isométrique, ou à Block Out ;
b. Déplacer le point de vue ;
c. Visualiser les nouvelles formes à venir ;
d. Gérer l'éclairage.

1.8 Casse briques en vue 3D

1.8.1 Descriptif général

Jouer à un casse brique, visuellement (principalement la vue 3D et les formes, pas nécessairement les effets visuels) proche de :
`https://youtu.be/pOnHz54DppI`

1.8.2 Fonctionnalités attendues

a. Pouvoir jouer à plusieurs niveaux du casse briques ;
b. Les niveaux peuvent être générés (optionel) ou modélisés (obligatoire) dans des fichiers externes au format texte et le format doit être documenté de manière à ce que chacun·e puisse en éditer de nouveaux facilement ;

c. Les niveaux doivent comporter différents types d'objets aux caractéristiques variées ;
d. Déplacer le point de vue ;
e. Modifier l'éclairage en fonction de certaines actions ;
f. Porter une attention particulière aux effets de lift.

1.9 FPS ou Doom-like – binôme possible

1.9.1 Descriptif général

FPS pour *First Person Shooter* (généralement traduit par jeu de tir subjectif - relativement à la vue) est un type de jeu vidéo dans lequel le joueur est immergé dans un environnement 3D virtuel où il incarne un personnage devant éliminer ses adversaires ; à cet effet il a, à sa disposition, une ou plusieurs armes (blanches ou à feu). Le terme Doom-like est souvent utilisé pour qualifier ce type de jeux et fait référence aux premiers succès du genre : Wolfenstein 3D, Doom et Quake.

1.9.2 Fonctionnalités attendues

a. Ce qui est attendu peut-être assez proche de ce qui est demandé pour le projet Pacman en mode FPS (cf. sous-section 1.3). Sinon lire les points suivants pour des variantes ou ajouts possibles ;
b. Réaliser un générateur de niveaux ;
c. Pouvoir se déplacer (donc détection de collisions) dans un environnement 3D de type labyrinthe avec sas et zones à accès restreint ; l'utilisateur pourra déverrouiller l'accès via mot de passe / énigmes / clés. L'ensemble de l'environnement est texturé, bumpé et éclairé. L'environnement (= un niveau) pourrait comporter des étages (escaliers et/ou ascenseurs) ;
d. Gérer plusieurs classes d'ennemis ayant chacune ses aptitudes et stratégies de combat.

1.10 Modéliser un ou plusieurs problèmes de physique

1.10.1 Descriptif général

En utilisant les éléments vus en cours – particulièrement le moteur de raster et de 3D – réaliser une simulation physique complexe comme : le problème circulaire restreint des trois corps (Poincaré) ; le double pendule et ses variantes ; un système de particules (pour simuler l'eau, la fumée, ...) ; ...

1.10.2 Fonctionnalités attendues

A discuter avec l'enseignant selon le choix de la simulation physique.

1.11 Randonnée – binôme possible

1.11.1 Descriptif général

L'idée est de proposer à l'utilisateur d'explorer un environnement naturel entièrement généré par ordinateur. Le joueur pourra, dans la mesure du possible et à pied, parcourir l'ensemble de ce monde virtuel.

1.11.2 Fonctionnalités attendues

a. Générer le terrain et le texturer (utilisation d'un algorithme de génération de MNT sur grille régulière ou import de données complètes ou complémentaires) ;
b. Texturer le terrain (bruit de Perlin pour le grain et coloration en fonction de l'altitude et des pentes) ;
c. Générer / placer des arbres (en mode filaire, voir projet L-Systèmes ?) ;
d. Se déplacer à pied en posant des contraintes sur les déplacements possibles (en fonction de la pente) ;
e. Créer un Skydome et le texturer avec du bruit de Perlin pour fabriquer des pseudo-nuages et gérer les conditions d'éclairage en fonction de l'heure (peut être temps-réel ou accélérée).

1.12 L-Systèmes

1.12.1 Descriptif général

Écrire un générateur de plantes basé sur les travaux d'Aristid Lindenmayer (Algorithmic Beauty of Plants, ouvrage disponible entièrement en ligne).

1.12.2 Fonctionnalités attendues

a. Écrire un générateur et un interprète géométrique de L-Systèmes (3D et au minimum des Bracketed L-Systems) ;
b. Importer plusieurs modèles depuis leur description donnée dans un fichier texte et pouvoir passer de l'un à l'autre ;
c. Se déplacer autour du L-Système, l'éclairer correctement ;
d. (Bonus) voir son évolution dans le temps.

1.13 Course tout terrains – binôme possible

1.13.1 Descriptif général

Ici, vous placez le joueur aux commandes d'un véhicule quelconque (deux ou trois hexahedrons réguliers et quatre tores texturés suffisent à la modélisation) où il devra effectuer une course dans laquelle il ralliera, sans autres contraintes que le relief, un point d'arrivée fixé sur la carte ; la course comprend la gestion des concurrents et se déroule dans un environnement naturel (reg, plaines ou petites collines). Le jeu est de type Arcade, la vue classique est "arrière".

1.13.2 Fonctionnalités attendues

a. Conduire le véhicule avec un modèle physique minimal ;
b. Pouvoir changer de point de vue : FPS (*First Person Shooter*), arrière ou haut-arrière ;
c. Détecter les collisions : principalement le sol et ses pentes mais aussi les véhicules des adversaires ;
d. Importer et modéliser le terrain à partir d'un MNT (une Height Map) ;
f. Avoir une vue orthogonale de la carte avec la position (ex. en vert) du joueur, (ex. en rouge) des concurrents et (ex. avec une croix) du point d'arrivée ;
g. Jouer à plusieurs via le réseau ;
h. L'IA contrôlera les concurrents.

1.14 Système solaire

1.14.1 Descriptif général

Modéliser le plus fidèlement possible le système solaire.

1.14.2 Fonctionnalités attendues

a. Modéliser et texturer le soleil, les planètes et leurs lunes ;
b. Pouvoir accélérer/décélérer le temps ;
c. Avoir des raccourcis pour visuliser en grand chaque astre et ses lunes quand il en a ;
d. Avoir des raccourcis pour visualiser l'orbite de chaque planète autour du soleil (de manière orthogonale au plan de l'orbite).

1.15 Combat aérien – binôme possible

1.15.1 Descriptif général

Placé dans la peau d'un aviateur, le joueur a pour mission de combattre tous les ennemis présents à l'écran. Le jeu place l'aéronef en vue arrière, il est de type Arcade, tout est géré comme sprite, seule le monde est 3D et on avance continuellement vers – plus ou moins – la même direction. Vous pouvez vous inspirer des exemples suivants :
Afterburner II (`https://youtu.be/65weTx0haog`) et Comanche 2.

1.15.2 Fonctionnalités attendues

a. Piloter un aéronef au dessus d'un relief simple mais texturé et sur lequel sont posés quelques sprites de décor ;
b. L'environnement doit être chargé (et déchargé) progressivement pour ne pas ralentir l'affichage (gestion de champs de vision maximal) ;
c. Combattre différents types d'ennemis ;
d. Détecter les collisions projectiles-aéronefs (joueur ou ennemis) et scorer.

1.16 Course acrobatique – binôme possible

1.16.1 Descriptif général

Ici, vous placez le joueur aux commandes d'un véhicule quelconque (modèle minimaliste) où il devra effectuer une course contre la montre. L'ensemble se déroule sur un circuit acrobatique (looping, rampes et sauts périlleux) et le joueur devra faire en sorte de terminer le circuit sans chuter. Le jeu est de type Arcade et vous pouvez vous inspirer de Stunt car (`https://youtu.be/q7w_0yP5RwU`).

1.16.2 Fonctionnalités attendues

a. Piloter un véhicule sur un circuit acrobatique ;
b. Pouvoir changer de point de vue : FPS (*First Person Shooter*), vue arrière, vue haut-arrière ;
c. Réaliser un mini système physique (au niveau des roues) et détecter les collisions ;
d. Éditer et/ou générer les circuits.

1.17 Course F1 / Karting – binôme possible

1.17.1 Descriptif général

Ici, vous placez le joueur aux commandes d'un véhicule (auto ou kart dont le modèle est très minimaliste) où il devra effectuer une course contre d'autres concurrents. Cette dernière se déroule sur un circuit et les coureurs devront effectuer un certain nombre de tours avant le finish. Le jeu peut être de type Simulation ou Arcade et vous pouvez vous inspirer de F1 Grand Prix, Vroom / Super Mario kart / Moto racer, etc.

1.17.2 Fonctionnalités attendues

a. Piloter sur un circuit en mode entraînement ou course ;
b. Pouvoir changer de point de vue : ras du sol, arrière, haut et points fixes sur le circuit ;
c. Détecter les collisions ;
d. Le programme devra contrôler les autres concurrents ;
e. Éditer et/ou générer des circuits ;
f. Jouer à plusieurs en splitant l'écran ou via le réseau.

1.18 Pong 3D

1.18.1 Descriptif général

Permettre à un joueur de jouer à un Pong en 3 dimensions. L'aire de jeu est délimitée par un tunnel et la raquette est déplaçable à la souris. Vous pouvez vous inspirer de la modeste mais efficace implémentation se trouvant à l'adresse :
`https://www.ponggame.org/3dpong.php` ou là
`https://youtu.be/pgUtluRO9e0`

1.18.2 Fonctionnalités attendues

a. Jouer a Pong en 3D et compter le score ;
c. Gérer des effets de lumières (éblouissement, éclairage du tunnel, etc.) ;
c. Gérer les effets sur la balle ;
d. Jouer contre un programme de plus en plus fort.

1.19 Rendu de fonctions mathématiques – binôme possible

1.19.1 Descriptif général

Écrire une application permettant à l'utilisateur de visualiser la courbe ou la surface de l'équation mathématique donnée en entrée. Vous pouvez vous inspirer du célèbre programme *gnuplot*.

1.19.2 Fonctionnalités attendues

a. Afficher la courbe / surface correspondant à une équation donnée en entrée ;
b. Gérer les pas et intervalles pour la visualisation ;
c. Se déplacer autour de la courbe / surface créée ;
d. Éclairer les surfaces générées : calculer les normales pour un résultat facetté / lissé ;
e. Introduire la variable temps et permettre ainsi la création d'animations.