



POLYTECH[®]
TOURS
Département
Aménagement et Environnement



université
de TOURS

CITERES
UMR 7324
Cités, Territoires,
Environnement et Sociétés

Equipe DATE
Dynamiques et Actions
Territoriales et
Environnementales

Projet de Fin d'Etudes

CityEngine : nouveau potentiel de modélisation 3D à l'usage des urbanistes ?



PRABEL Gaétan

2017-2018

Directeur de recherche
LARRIBE Sebastien

CityEngine : nouveau potentiel de modélisation 3D à l'usage des urbanistes ?



Tuteur : LARRIBE Sebastien
Année 2018

Auteur : Gaétan PRABEL

Avertissement

Cette recherche a fait appel à des lectures, enquêtes et interviews. Tout emprunt à des contenus d'interviews, des écrits autres que strictement personnel, toute reproduction et citation, font systématiquement l'objet d'un référencement.

L'auteur (les auteurs) de cette recherche a (ont) signé une attestation sur l'honneur de non plagiat.

Formation par la recherche, Projet de Fin d'Etudes en génie de l'aménagement et de l'environnement

La formation au génie de l'aménagement et de l'environnement, assurée par le département aménagement et environnement de l'Ecole Polytechnique de l'Université de Tours, associe dans le champ de l'urbanisme, de l'aménagement des espaces fortement à faiblement anthropisés, l'acquisition de connaissances fondamentales, l'acquisition de techniques et de savoir-faire, la formation à la pratique professionnelle et la formation par la recherche. Cette dernière ne vise pas à former les seuls futurs élèves désireux de prolonger leur formation par les études doctorales, mais tout en ouvrant à cette voie, elle vise tout d'abord à favoriser la capacité des futurs ingénieurs à :

- Accroître leurs compétences en matière de pratique professionnelle par la mobilisation de connaissances et de techniques, dont les fondements et contenus ont été explorés le plus finement possible afin d'en assurer une bonne maîtrise intellectuelle et pratique,
- Accroître la capacité des ingénieurs en génie de l'aménagement et de l'environnement à innover tant en matière de méthodes que d'outils, mobilisables pour affronter et résoudre les problèmes complexes posés par l'organisation et la gestion des espaces.

La formation par la recherche inclut un exercice individuel de recherche, le projet de fin d'études (P.F.E.), situé en dernière année de formation des élèves ingénieurs. Cet exercice correspond à un stage d'une durée minimum de trois mois, en laboratoire de recherche, principalement au sein de l'équipe Ingénierie du Projet d'Aménagement, Paysage et Environnement de l'UMR 6173 CITERES à laquelle appartiennent les enseignants-chercheurs du département aménagement.

Le travail de recherche, dont l'objectif de base est d'acquérir une compétence méthodologique en matière de recherche, doit répondre à l'un des deux grands objectifs :

- Développer toute ou partie d'une méthode ou d'un outil nouveau permettant le traitement innovant d'un problème d'aménagement
- Approfondir les connaissances de base pour mieux affronter une question complexe en matière d'aménagement.

Afin de valoriser ce travail de recherche nous avons décidé de mettre en ligne sur la base du Système Universitaire de Documentation (SUDOC), les mémoires à partir de la mention bien.

Sommaire

Avertissement.....	5
Formation par la recherche, Projet de Fin d'Etudes en génie de l'aménagement et de l'environnement	5
Sommaire.....	6
Introduction.....	7
1) Une prise en charge plus commode du Modèle Numérique de Terrain sur CityEngine	8
a) Précision du Modèle Numérique de Terrain	8
b) Import du Modèle Numérique de Terrain	9
c) Positionnement des aménagements urbains sur le Modèle Numérique de Terrain.....	11
2) CityEngine : un système d'extrusion et texturage programmable des objets 3D.....	13
3) CityEngine : Une meilleure interopérabilité pour des rendus plus variés et qualitatifs.....	14
a) Comparaison globale des fonctionnalités d'export.....	14
b) Deux exemples de rendus possibles à partir d'un modèle CityEngine.....	16
4) Le prix à payer de CityEngine : la licence et les ressources matérielles nécessaires.....	18
Conclusion.....	20
Remerciements.....	21
Sources.....	21
Annexe 1: Codes des règles utilisés pour la modélisation de Charleval dans CityEngine.....	22
Annexe 2 : Protocole de création et d'application de nouvelles textures à partir des règles présentes en annexe 1.....	36
Annexe 3 : Plus de détails sur l'interopérabilité de SketchUp et CityEngine.....	37
Annexe 4: Paramètres d'export depuis CityEngine pour les logiciels Unity3D et LumenRT.....	40
Annexe 5 : Paramètres nécessaires au fonctionnement d'une application de réalité virtuelle sur Android dans Unity3D.....	41

Introduction

L'atelier du second semestre de DAE4 permet aux étudiants de l'option RESEAU de faire des propositions sur une commande réelle effectuée par un élu, généralement un quartier à concevoir. Par groupes, les étudiants conçoivent leur projet en se répartissant les différentes thématiques à considérer : les logements, la production énergétique, les systèmes de transports, les consommations d'eau, ... Ils doivent produire deux types de rendus :

- Un bilan global énergétique de leur quartier, en considérant la consommation et la production d'énergie réalisée,
- Une modélisation 3D de leur projet, permettant de rendre compte de la mise en œuvre spatiale des propositions du groupe.

Pour cette dernière, les étudiants utilisent le logiciel SketchUp pour modéliser leur nouveau quartier : c'est un logiciel gratuit, facile à prendre en main, et qui permet toutes les possibilités de design pour les formes urbaines envisagées dans le projet. En parallèle, à l'aide des bases de données géographiques disponibles pour la zone étudiée, ils réalisent un modèle de l'environnement autour du site du projet, à l'aide d'un logiciel SIG comme ArcGIS qui est importé dans SketchUp et leur permet ensuite d'incorporer leur nouveau quartier. Afin de pouvoir les communiquer au commanditaire, ils doivent alors réaliser des photos et vidéos ou animations de leur projet. Mais, à la base, SketchUp est un logiciel plutôt destiné aux architectes, donc pour des projets à l'échelle du bâtiment. L'environnement du site du projet comprend plusieurs dizaines de bâtiments, ce qui amène à s'interroger de l'intérêt de SketchUp pour cette modélisation.



Figure 1 : Logo du logiciel SketchUp
Source : <https://seeklogo.com/vector-logo/282937/sketchup>

CityEngine, logiciel propriétaire d'ESRI (la même société commercialisant la suite ArcGIS) est, comme son nom l'indique, un « moteur de villes ». Il apparaît donc que cette solution est conçue pour gérer des modèles de quartiers de villes, comprenant un nombre important de bâtiments. Ainsi, la question de la pertinence de l'utilisation de ce logiciel se pose pour la modélisation de l'environnement autour du quartier.



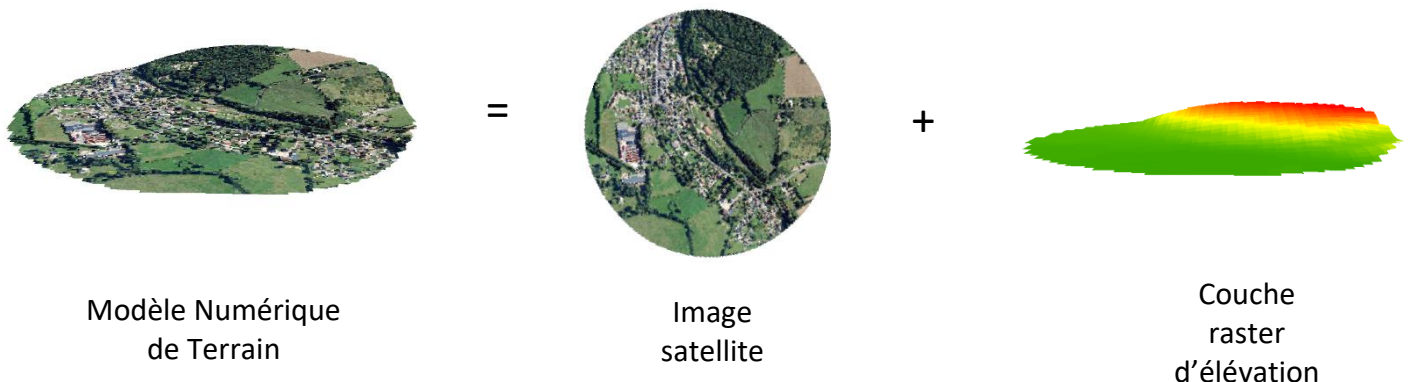
Figure 2 : Logo du logiciel CityEngine
Source : <https://store.esriuk.com/products/#business>

Ce projet a donc pour but de comparer SketchUp et CityEngine sur leur usage pratique pour la modélisation de cet environnement, en considérant 4 aspects :

- D'abord, leur prise en charge du Modèle Numérique de Terrain du site considéré ;
- Puis, l'application de textures sur les formes géométriques (bâtiments, routes) présentes autour du site ;
- Ensuite, l'interopérabilité des 2 logiciels, conditionnant la qualité des rendus à disposition du commanditaire ;
- Enfin, les coûts nécessaires à l'acquisition et au fonctionnement des 2 solutions, permettant une projection financière sur le déploiement de ces dernières avant leur utilisation par les étudiants.

1) Une prise en charge plus commode du modèle numérique de terrain sur CityEngine

Dans la construction d'un modèle urbain en 3 dimensions, la première étape est la construction d'un modèle numérique de terrain de la zone considérée, sur lequel s'ajouteront les différentes formes urbaines, entre autres les bâtiments et réseaux de transport. Généralement, l'élaboration d'un modèle numérique s'effectue au départ dans un logiciel de SIG 3D, type ArcScene. Il se compose d'une orthophoto (image satellite) raster de la zone considérée (provenant de la BD Ortho en France), qui est alignée sur le relief du terrain, codée par une couche raster d'élévation du terrain mise en 3 dimensions et provenant de la BD Alti du département considéré.



*Figure : Schéma de composition d'un Modèle Numérique de Terrain
Source : Gaétan Prabel*

a) Précision du Modèle Numérique de Terrain

SketchUp n'offre pas la possibilité de générer des Modèles Numériques de Terrain. Ainsi, la qualité et la précision du modèle rendu dépend de la source de la donnée traitée dans ArcGIS avant l'import. Ainsi, pour la zone de Charleval, la seule base de données nationale exploitable a une résolution de 25 m, c'est-à-dire que chaque pixel est un carré de 25 m de côté. Cette grande résolution entraîne des pentes anormalement élevées sur le modèle, qui ne se retrouvent pas dans la réalité.

Depuis la version 2016.0, CityEngine offre la possibilité de télécharger des données d'orthophoto et d'altitude sur une zone sélectionnée depuis la base d'Esri World Elevation, grâce à l'outil « Get Map Data ». Moyennant l'inscription gratuite à une version d'essai d'ArcGIS Online (valable 21 jours), elle permet l'acquisition de données d'altitudes et d'images satellites jusqu'à la résolution 4K, soit 4096 pixels sur un côté de carré d'1 km. Ainsi, la résolution de ce raster d'altitude est de 25cm, donc 100

fois plus petite que la BD Alti utilisée. La précision en est donc grandement améliorée, et les phénomènes de pentes anormalement élevées sont très largement réduits.

On remarquera que la résolution obtenue est plus petite que n'importe quelle base de données altimétriques française, car la résolution la plus petite de la BD Alti en France est 50 cm.

Par ailleurs, le même outil permet également de télécharger le réseau routier et le bâti de la zone sélectionnée provenant de la base de données d'OpenStreetMap en cochant les cases « Download Networks » et « Download Polygons » dans l'utilitaire de téléchargement de données ci-contre.

Si cet outil néglige quelque peu l'apprentissage de la SIG, il permet l'obtention beaucoup plus rapide de toutes ses données. En effet, sur ArcGIS, il est nécessaire de couper toutes les bases de données (Ortho, Topo Bâti et Routes, Alti) du département souhaité pour ne sélectionner que les éléments situés dans la zone d'étude, avant l'importation dans SketchUp et/ou CityEngine.

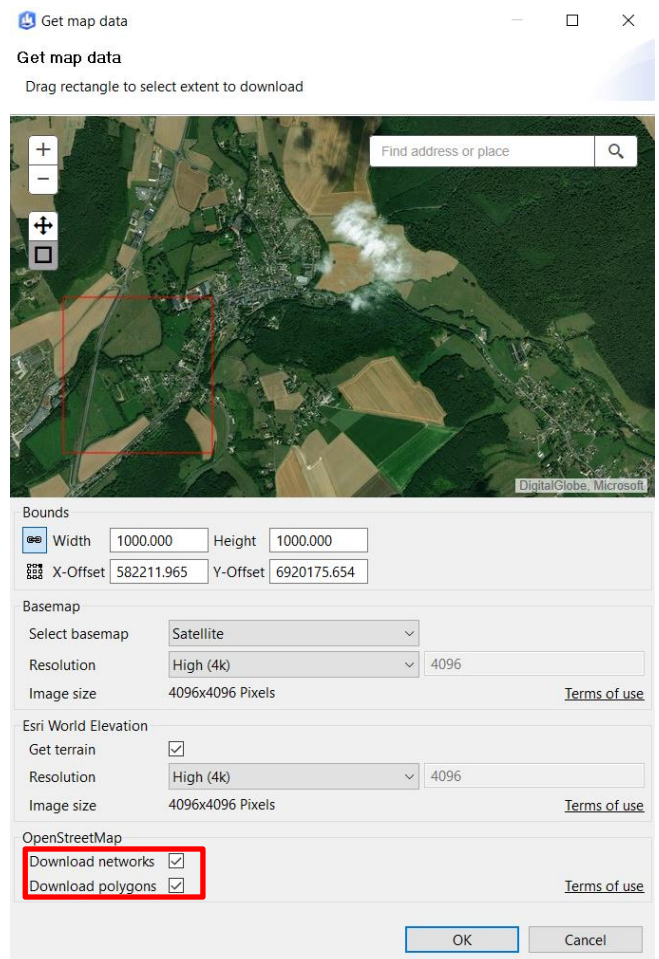


Figure 3 : Fenêtre de téléchargement de données cartographiques dans CityEngine
Source : Gaétan Prabel

b) Import du Modèle Numérique de Terrain

Même si cet outil n'est pas utilisé pour le chargement des données, l'utilisateur bénéficiera quand même de l'import dans CityEngine. En effet, l'absence de format directement compatible entre SketchUp et des logiciels de SIG 3D comme ArcScene impose le passage par un logiciel tiers, tel que Toaster Data ou Blender, selon le schéma suivant :

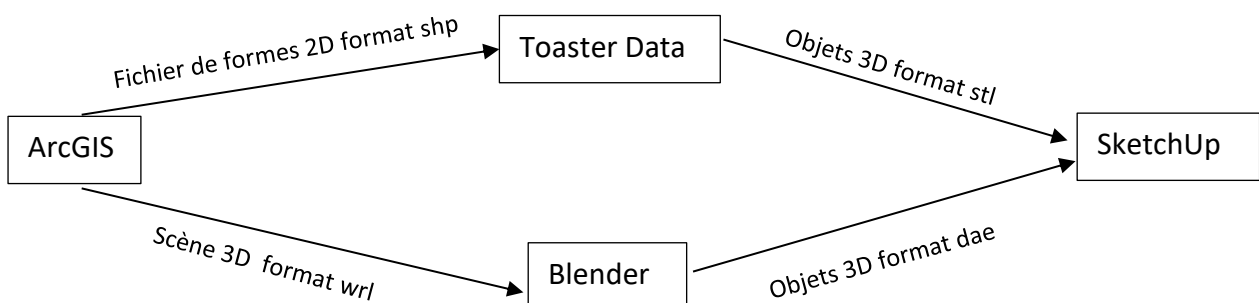


Figure 4 : Schéma des possibilités d'import du Modèle Numérique de Terrain sous SketchUp
Source : Gaétan Prabel

Les deux formats importés (stl et dae) ne prennent pas en charge la texture du modèle, si bien que l'image satellite n'apparaît pas dans le modèle 3D de SketchUp, remplacé par une couleur uniforme, comme le montre le modèle suivant :

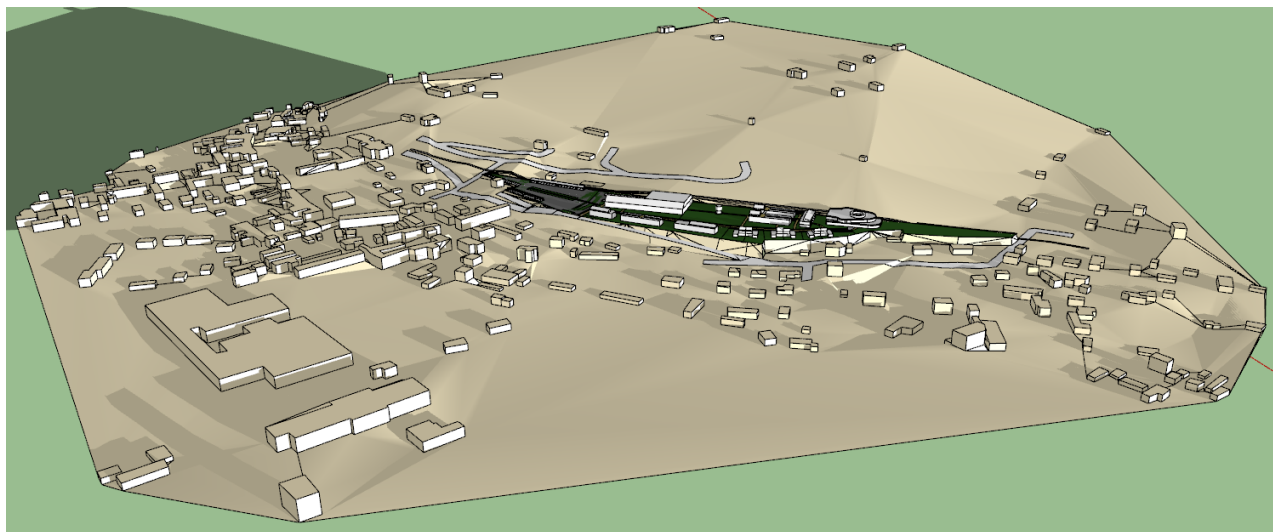


Figure 5 : Modèle numérique sous le logiciel Sketchup
Source : Atelier DAE4 E conception 2017

A l'inverse, CityEngine a été intégré à la suite ArcGIS Desktop, et donc il prend en charge tous types de données SIG. En particulier, il est possible d'importer un terrain en sélectionnant le fichier raster de l'image satellite et celui de l'altimétrie (aux formats tif, jpg, png,...), permettant d'obtenir directement le Modèle Numérique de Terrain en 3 dimensions, intégrant donc en texture l'orthophoto du site considéré.

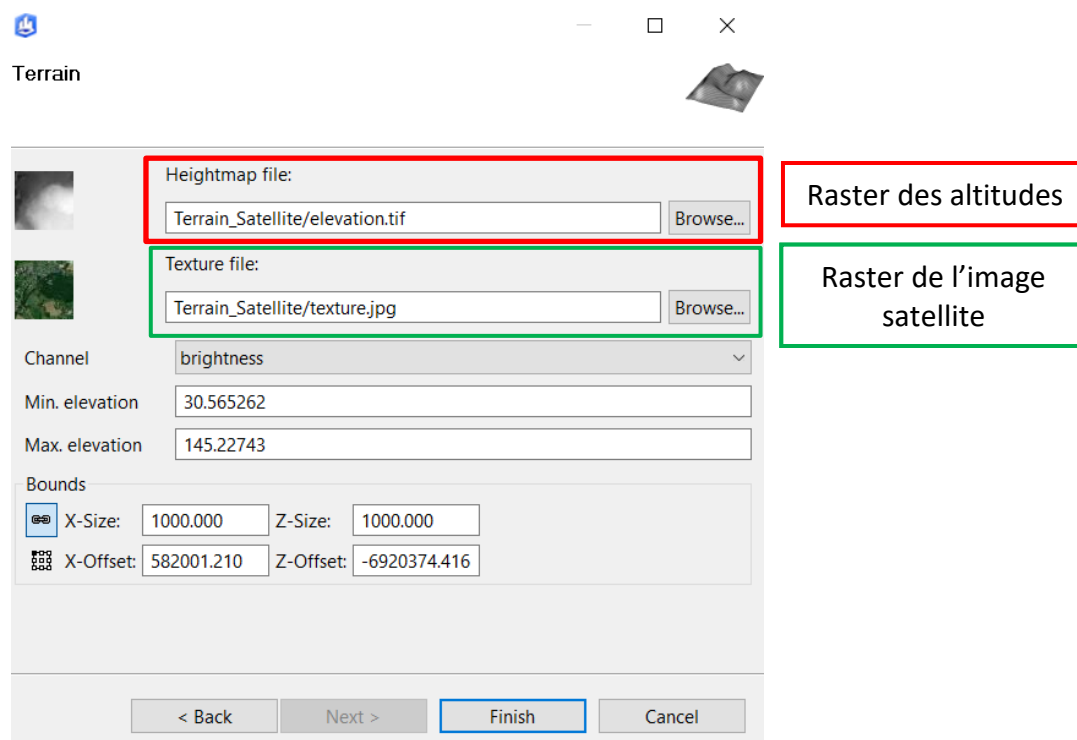


Figure 6 : Import de Modèle Numérique de Terrain dans CityEngine
Source : Gaétan Prabel

Comme le montre l'image ci-dessous, cette visualisation de l'image satellite permet de donner du contexte au modèle, offre une vision plus réaliste du site, et permet à l'utilisateur de s'y projeter plus facilement.

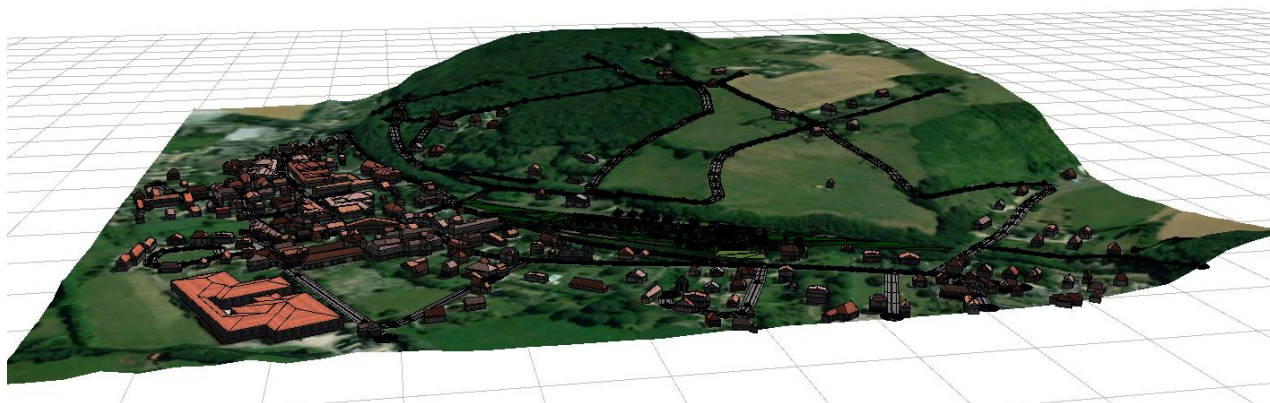


Figure 7 : Modèle numérique sous le logiciel CityEngine
Source : Gaétan Prabel

Bien évidemment, cette partie du travail peut être évitée si l'utilisateur télécharge les données directement depuis CityEngine (voir partie a.).

c) Positionnement des aménagements urbains sur le Modèle Numérique de Terrain

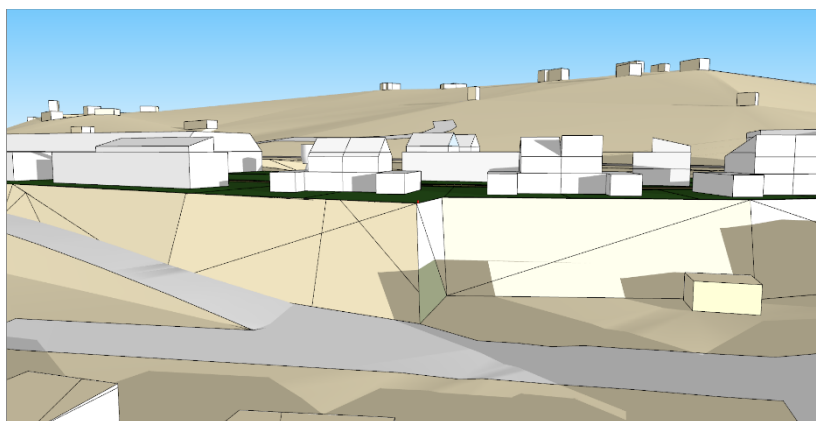
Une fois le modèle importé, les aménagements urbains (routes, bâtiments) peuvent être ajoutés sur le modèle : l'environnement proche du projet d'une part, et la modélisation du projet envisagé d'autre part.

Concernant le quartier autour du projet, en utilisant SketchUp, les bases de données géographiques des routes et bâtiments doivent être importées de la même manière que le Modèle Numérique de Terrain, en utilisant un logiciel tiers. Ce protocole fait perdre la composante géoréférencée de ces formes, obligeant à un positionnement manuel sur le site. Cette démarche apparaît peu commode car le modèle ne comprend pas de texture d'image satellitaire, compliquant la prise de repères. Au contraire, CityEngine prend directement en charge ce type de base de données, au format géodatabase ou fichier de formes. Ainsi, si le système de coordonnées défini dans le modèle CityEngine est le même que celui utilisé dans les fichiers importés, le positionnement géographique sera automatiquement réalisé. De plus, l'alignement des entités géométriques par rapport au terrain s'effectue facilement après sélection des entités :

- Par l'outil « Align Graph to Terrain » pour les entités de type réseau, comme les routes,
- Par l'outil « Align Shapes to Terrain » pour les autres entités comme les bâtiments.

D'autre part, le modèle du projet réalisé par les groupes d'atelier sera généralement réalisé sur SketchUp, indépendamment du modèle de terrain, sur fond plat. Si le modèle d'ensemble est réalisé sur ce même logiciel, le modèle de base n'est pas précis (voir partie a), il faut donc combler les pentes anormalement élevées présentes sur le terrain, qui n'existent pas dans la réalité. L'utilisateur doit construire un socle plat manuellement, à l'aide de nouvelles lignes et coupures du modèle de terrain comme montré dans l'image ci-dessous, une tâche très fastidieuse pouvant représenter plusieurs dizaines de minutes de travail.

Modèle du projet
importé



Socle construit à
partir du modèle
de départ

Figure 8 : Construction du socle de base pour le modèle du projet sous Sketchup

Source : Atelier DAE4 E conception 2017

En intégrant le modèle sur CityEngine, après l'avoir géopositionné sur SketchUp, il lui suffit de l'importer au format « kmz » et de l'aligner sur le terrain déjà construit avec l'outil « Align Terrain to Shapes », ce qui lui permet de gagner beaucoup de temps. Le résultat apparaît également beaucoup plus fluide, comme le montre l'image ci-dessous :

Modèle du projet
importé



Terrain aligné
automatiquement,
aspect plus fluide

Figure 9 : Alignement du modèle du projet dans CityEngine

Source : Gaétan Prabel

L'intérêt d'un modèle de terrain en 3 dimensions réside dans l'impression de réalisme qu'il peut dégager. Pour cela, les deux logiciels proposent chacun une façon de donner une texture imitant les matériaux de construction. Ainsi colorées, les faces des bâtiments et linéaires de routes apparaissent plus attrayants, et permettent une meilleure immersion pour les observateurs du modèle.

2) CityEngine : un système d'extrusion et texturage programmable des objets 3D

Dans le logiciel SketchUp, une texture choisie par l'utilisateur est appliquée sur les faces sélectionnées. Etant donné le nombre important de bâtiments dans un tel modèle (environ 250 bâtiments), ce procédé apparaît fastidieux pour lui. Par exemple, au sein d'une sélection de plusieurs bâtiments, aucune distinction n'est faite entre les faces des toits et celles des murs, et donc la même texture sera appliquée partout, alors que les matériaux utilisés sont différents entre façades et toits. De plus, la hauteur d'extrusion est fixée par le logiciel SIG d'où sont importés les bâtiments, et elle ne peut être modifiée dans SketchUp. Ainsi, par économie de temps, les bâtiments ont été laissés dans leur état d'origine lors de l'atelier d'écoconception.

Pour ce qui est de CityEngine, le procédé d'extrusion et de texturage des bâtiments repose sur une application automatisée d'un programme au format cga, appelé règle. Ce programme applique une texture sur les surfaces des bâtiments et des routes, ainsi qu'une extrusion sur une hauteur déterminée. Il fait appel à d'autres règles, régissant séparément les textures des toits et des façades des bâtiments, comme expliqué dans les codes en annexe 1. Ainsi, il est possible de régler des variables comme la hauteur d'extrusion, la pente des toits, la largeur des routes, ... en utilisant entre autres les données issues des tables attributaires des formes associées. Par exemple, la hauteur d'extrusion peut être définie comme étant la valeur du champ « Hauteur » de la table attributaire des bâtiments de la BD Topo. C'est le cas aussi pour la largeur et le nombre de voies des routes.

L'intérêt premier de ce fonctionnement est le choix parmi un très large éventail de textures pour les façades et les toits. En effet, ces dernières sont appliquées à partir d'images représentant la texture voulue, et intégrées dans un dossier dont le code précise le chemin d'accès. Avec la quantité d'images proposées sur Internet, le choix est immense. Et pour se rapprocher davantage encore de la réalité, des photos réelles prises sur le terrain et redimensionnées peuvent être utilisées. L'autre avantage est le fait que ces règles peuvent se répliquer sans tout connaître à leur syntaxe propre : pour créer une nouvelle règle en partant des exemples de programmes donnés en annexe, il suffit de copier l'ensemble du code pour le type de faces voulu (toits, façades, murs ou même route) et de changer les chemins d'accès correspondant aux images et aux règles. L'annexe 2 donne plus d'informations sur le protocole pour créer et assigner une nouvelle règle à une sélection de bâtiments.

Après application de ces règles, on obtient des rendus de bâtiments tels que proposés sur l'image suivante :



Figure 10 : Image du modèle CityEngine montrant les textures appliquées aux bâtiments et routes
Source : Gaétan Prabel.

Des modèles de ville comme ceux réalisés dans SketchUp ou CityEngine n'ont pas forcément vocation à être le rendu final d'un projet. Ainsi, la possibilité d'intégrer une solution logicielle dans une ensemble de traitements informatiques aboutissant à des rendus finaux variés constitue aussi un argument permettant de justifier son utilisation.

3) CityEngine : Une meilleure interopérabilité pour des rendus plus variés et qualitatifs

a) Comparaison globale des fonctionnalités d'export

Dans cette partie, nous allons comparer les possibilités d'exports de fichier depuis SketchUp et CityEngine, en classifiant les logiciels utilisables en différents groupes :

- Les logiciels de CAO/DAO (Conception Assistée par Ordinateur / Dessin Assisté par Ordinateur) tels que AutoCAD, permettent un rendu du modèle type dessin technique, incluant les dimensions du projet.
- Les logiciels de traitement d'image, tels qu'Adobe Photoshop, permettent des rendus professionnels de type image 2D.
- Les logiciels d'imprimerie 3D, tels que TinkerCAD, sont des logiciels communiquant avec des imprimantes 3D et permettant l'impression des modèles à l'échelle.
- Les logiciels SIG (Systèmes d'Informations Géographiques) permettent de visualiser des données 2D et 3D, et d'effectuer des analyses et traitements spatiaux. On y trouve des logiciels gratuits comme Google Earth ou plus complets comme la suite ArcGIS for Desktop.

- Les moteurs de jeu vidéo (comme Unity et Unreal Engine) sont des logiciels intégrant des moteurs graphiques et physiques permettant la construction d'applications animées en 3D du type jeux vidéo, ou encore l'utilisation de la réalité virtuelle.
- Les logiciels d'infographie 3D, comme Maya ou 3DS Max, permettent la génération de scènes numériques 3D du projet, impliquant entre autres le positionnement du modèle importé ou les réglages des lumières et de la simulation des phénomènes physiques.
- Les plateformes Web, dans lesquels le projet est stocké pour être affiché et utilisé sur des sites Internet, notamment la plateforme ArcGIS online, pour afficher des modèles de villes en 3 dimensions.
- Les interfaces de programmation, permettant à l'utilisateur de modifier directement sur le code informatique générant les formes et textures du projet.

L'interopérabilité avec ces différents logiciels sera comparée dans le tableau ci-dessous, avec des notes comprises entre 0 et 2 croix, 0 indiquant la non-compatibilité des logiciels entre eux, et 2 indiquant de très bonnes relations. CityEngine étant un logiciel payant, la licence Pro de SketchUp, incluant plus de fonctionnalités, a été ajoutée à la comparaison avec la version gratuite pour plus d'équité. La justification des résultats est proposée en annexe 3.

Type de logiciel	SketchUp	SketchUp Pro	CityEngine
Logiciel de CAO/DAO		XX	X
Logiciel de traitement d'image	X	X	XX
Logiciel d'imprimerie 3D	X	X	
Logiciel SIG	X	X	XX
Moteur de jeux vidéo		X	X
Logiciels d'infographie 3D		X	XX
Plateforme Web			X
Interface de programmation			X

Tableau 1 : Evaluation de l'interopérabilité de SketchUp et CityEngine vers différents types de logiciels

Source : Gaétan Prabel

Ce tableau montre les limites de la version gratuite de SketchUp, car elle n'est principalement pas compatible avec les logiciels de CAO/DAO, infographie 3D et Internet. La version Pro corrige une partie de ces manques, avec une forte interconnexion avec les logiciels de CAO/DAO (par le support du format natif .dwg d'AutocAD), et permettant l'export vers les moteurs de jeux vidéo et d'infographie 3D. Bien qu'elle ne soit pas directement compatible avec les logiciels d'impression 3D, CityEngine supporte mieux les logiciels de traitement d'image, grâce au format Pixar, utilisé dans les applications graphiques liées au rendu d'images 3D. De même, il s'intègre mieux aux logiciels de

SIG, notamment la suite ArcGIS, grâce à l'export de géodatabases. Par ailleurs, par rapport à SketchUp Pro, il ajoute une compatibilité avec quelques extensions supplémentaires pour des logiciels d'infographie 3D spécifiquement utilisés pour réaliser des rendus de modèles urbains, comme LumenRT. Enfin, il est le seul des 3 logiciels comparés à permettre un export de Scène Web visualisable dans ArcGIS online, à condition d'avoir souscrit un compte payant.

Comme expliqué précédemment, ces différents supports permettent l'accès à différents rendus accessibles depuis un projet CityEngine, dont nous allons voir deux exemples dans la partie suivante.

b) Deux exemples de rendus possibles à partir d'un modèle CityEngine

- Rendu photoréaliste avec LumenRT

Des rendus de type photo ou vidéo d'un projet urbain permettent au commanditaire de se projeter dans son potentiel aménagement. Dans l'atelier d'écoconception de DAE4, avec un rendu du modèle sur SketchUp, les vidéos de démonstration du projet ont été réalisées grâce à l'outil intégré à ce logiciel, tandis que les images ont été rendues grâce au logiciel Kerkithea. SketchUp offre un rendu avec un réalisme limité, ce qui se remarque dans la qualité des images rendues, comme on peut le voir dans l'image suivante :



Figure 11 : Exemple de rendu image créée avec Kerkithea
Source : Atelier Econception DAE4 2017

Utiliser CityEngine pour la modélisation du quartier permet l'export vers un logiciel de rendu « photoréaliste », LumenRT, avec la compatibilité du format EON (avec les paramètres donnés en annexe 4). D'abord, Il offre un rendu beaucoup plus qualitatif que SketchUp, grâce à plus de possibilités de réglages comme la gestion des nuages au ciel, le rendu des espaces éclairés par le soleil et des ombres portées des bâtiments, ainsi que la simulation des feuilles soufflées par le vent. De plus, il permet un rendu du projet intégré dans son environnement, profitant d'un des avantages de CityEngine. Enfin, les rendus photo et vidéo sont réalisables dans le même logiciel, offrant une continuité dans les différents rendus du projet. Voici un exemple de rendu image sur le modèle de Charleval :



*Figure 12 : Exemple de rendu image crée avec LumenRT
Source : Gaétan Prabel*

- Rendu du modèle en réalité virtuelle avec le logiciel Unity3D

Un autre intérêt de CityEngine est sa compatibilité avec Unity3D, un moteur de jeu vidéo, par un export au format FBX, avec les paramètres donnés en annexe 4. Unity3D permet la construction d'applications utilisant la réalité virtuelle, sur différentes plateformes : Windows, Android, iOS, entre autres. L'application construite peut être utilisée avec différentes plateformes : Oculus (compatible avec les casques HTC Vive, Oculus Rift et Samsung Gear VR), et sur Android, via une application au format apk, Google Cardboard (compatible à partir de la version 4.4 d'Android) et Daydream (à partir de la version 7.0 d'Android). L'annexe 5 précise les paramètres nécessaires à son fonctionnement sous Android. La réalité virtuelle facilite encore la projection du projet sur le terrain en proposant une nouvelle expérience d'immersion pour le commanditaire.



Figure 13 : Plateformes compatibles avec les applications en réalité virtuelle d'Unity :
à gauche, le casque Samsung Gear VR ; à droite, la Google Cardboard

Sources : <https://www.lesnumeriques.com/casque-realite-virtuelle/samsung-gear-vr-2015-sm-322-p28793/test.html>
https://store.google.com/product/google_cardboard

Mais si cette suite logicielle a de nombreux avantages, vus dans les paragraphes précédents, ils se paient sur la licence et les ressources matérielles nécessaires pour la faire fonctionner.

4) Le prix à payer de CityEngine : la licence et les ressources matérielles nécessaires

L'utilisation de CityEngine entraîne des surcoûts financiers non négligeables pour le modelleur, comparé à SketchUp : d'abord, sa licence est payante, et en plus, il demande plus de ressources matérielles sur son ordinateur pour fonctionner correctement. Tous ces surcoûts sont résumés dans le tableau suivant. Les suggestions correspondent aux configurations minimales recommandées par les éditeurs des logiciels, et les prix indiqués sont ceux du composant requis le moins cher affiché sur le site LDLC en mars 2018 :

Postes budgétaires	SketchUp	CityEngine
Mémoire Vive	4 Go (51.95€ pour de la DDR4 cadencée à 2400MHz)	16 Go (196.95€ pour de la DDR4 cadencée à 2400MHz)
Carte graphique	Carte graphique compatible OpenGL 3.0 avec 512 Mo de mémoire dédiée (37.94€ pour une AMD Radeon HD 6450)	Carte graphique compatible OpenGL 4.1 avec 2 Go de mémoire dédiée (49.99€ pour une Nvidia GT710)
Prix de la licence	0€ pour la version Make 565€ pour la version Pro	900€ pour la version Basic 7220€ pour la version Advanced (utilisée dans ce projet)
SURCOUT TOTAL	89.89€ pour la version Make 654.89€ pour la version Pro	1146.94€ pour la version Basic 7466.94€ pour la version Advanced

Tableau 2 : Surcoûts liés à l'implantation des 2 solutions logicielles
Source : Gaétan Prabel

Ce tableau montre que la solution CityEngine est beaucoup plus chère que SketchUp, en particulier avec la licence Advanced utilisée dans ce projet. Le surcoût est plus de 10 fois supérieur à celui de la plus onéreuse des versions de SketchUp !

En ce qui concerne les logiciels présentés dans la partie précédente, Unity3D dispose d'une version gratuite et ne demande pas de ressources matérielles supplémentaires. Quant à LumenRT, sa licence est payante, et il demande une carte graphique plus performante que la Nvidia GT710 recommandée dans le tableau ci-dessus.

Conclusion :

En guise de conclusion, le tableau de synthèse suivant résume les avantages et inconvénients de chacun des 2 logiciels :

Logiciels	SketchUp	CityEngine
Avantages	Un design à l'infini à l'échelle du bâtiment Existence d'une version gratuite Aucune connaissance technique nécessaire pour le prendre en main Faibles exigences en ressources matérielles	Support complet des Modèles Numériques de Terrain et des données SIG Comprend un outil de téléchargement de données SIG Application automatisée et aisée de textures sur des entités géométriques à l'aide de règles Bonne interopérabilité vers des logiciels de nature différente
Inconvénients	Support incomplet des Modèles Numériques de Terrain Ne prend pas en charge directement les données SIG Application manuelle et fastidieuse des textures sur les formes géométriques Interopérabilité limitée sur la version gratuite	Prix de la licence très élevé Plus gourmand en ressources matérielles

Tableau 3 : Avantages et Inconvénients de SketchUp et CityEngine

Source : Gaétan Prabel

Au prix d'un certain surcoût financier, CityEngine permet ainsi de combler les limites de SketchUp en matière de prise en charge des Modèles Numériques de Terrain et des données SIG, et d'automatisation de l'application des textures sur les formes géométriques. Etant un logiciel à visée plus professionnelle que SketchUp, il offre des possibilités d'export vers un plus grand nombre de logiciels, ouvrant la porte à des rendus plus divers (photos, vidéos, animations, réalité virtuelle), plus qualitatifs et plus réalistes. Ce type de rendu constitue une aide à la décision pour le commanditaire du projet : plus il est réaliste et immerge l'élus dans son terrain, plus il se projettera facilement sur les propositions réalisées par les étudiants de l'atelier.

Personnellement, ce projet m'aura permis de construire une suite d'opérations permettant, à partir d'un modèle SketchUp de projet et de données SIG, d'aboutir à des rendus divers et variés, et

constitue aussi ma première expérience de réalité virtuelle. Ayant été confronté à des logiciels de nature variée (logiciel SIG, modeleur 3D CityEngine, moteur de jeu vidéo, logiciel de rendu 3D), il m'aura aussi permis de développer mes compétences en informatique.

Remerciements :

Je tiens à remercier M. Sebastien LARRIBE, le tuteur de ce projet, pour sa disponibilité, les conseils précieux qu'il a pu m'apporter, ainsi que pour la mise à disposition du matériel nécessaire pour tester le résultat du projet, notamment la réalité virtuelle.

Sources :

Chaîne Youtube "CityEngine Essential Skills" contenant des tutoriaux sur les outils de base de CityEngine :

https://www.youtube.com/watch?v=RHKU2owulKU&list=PLWGP11THb9EpvmuokMrj_yJ6L4qCuKVli

Tutorial décrivant certains paramètres de Unity pour une application en réalité virtuelle sur Android :

<https://developers.google.com/vr/develop/unity/get-started>

Annexe 1 : Codes des règles utilisés pour la modélisation de Charleval dans CityEngine

1) Règle permettant le texturage des routes :

```
/**
 * File:      Street Construction Simple.cga
 * Created: 1 April 2014
 * Author:   Esri R&D Center Zurich
 */

version "2016.1"
#####
# initial variables from attribute table

attr LARGEUR = 0
attr NB_VOIES = 0
attr SENS = ""
attr Shape_Leng = 0

#####
# Control attributes

@Order(1) @Range(0,4)
attr NbrOfRightLanes = _getInitialRightLanes
@Order(2) @Range("yellow","white","none")
attr Centerline = "white"

# Typically, the width values are overridden by the shape parameters, however in
# case no shape parameters are available (e.g. on
# Junction or Freeway shapes of nodes), the texture
# coordinates contain the needed values
@Hidden @Order(3) @Range(3.5,30)
attr streetWidth = LARGEUR
# REALWORLD-distance in V-direction
# corresponds to width of street (in case the
# geometry does not contain rounded entry geometry)
@Hidden @Order(4) @Range(1,6)
attr laneWidth = LARGEUR/NB_VOIES # note that TEXTURE-
# distance in V-direction corresponds to number of lanes (as generated by
# CityEngine)

const TextureFolder = "Streets/"
const SidewalkHeight = 0.2
#
height of sidewalk (and depth of curbs)

const nLanesTotal = NB_VOIES
const oneWay = SENS == "Direct" || SENS == "INVERSE"

_getInitialRightLanes = case nLanesTotal>2: rint(nLanesTotal/2+0.01)
                        else : 30%
rint(nLanesTotal/2) 40%: nLanesTotal else: 0 # adding some randomness to get a
more diversified default appeareance

# RULES

@StartRule
Street --> # split away the side geometry on the streets
```

Initialisation des valeurs de la table attributaire utilisées dans le programme

Utilisation de l'attribut « Largeur » de la table attributaire des Routes de la BD TOPO pour extruder la largeur de la route texturée, gérée aussi par l'attribut « streetWidth »

Dossier des « Assets » contenant les images des textures utilisés pour construire les routes

```

        split(v,uvSpace,0){ -geometry.vMin: Asphalt          # the lanes start at
v-coord 0 i.e. everything below can be splitted away (= asphalt)
                                | nLanesTotal      : Lanes      # the
lanes end at v-coord nLanesTotal
                                | ~1              : Asphalt }   #
all remaining geometry beyond v-coord nLanesTotal can be split away

Lanes -->
    case Centerline=="none" || oneWay:                # LANE WITHOUT CENTERLINE:
these textures can be repeated also in v-direction, i.e. every lane contains ONE
edge line on the texture
        scaleUV(0,1,nLanesTotal*256/(nLanesTotal*256+18))    # v-
coord: texture image has 4 lanes (see above*)... and getting rid of the
repeating edge line by cutting away a part of the texture (one lane is 256 px in
the texture and a line is 18 px wide))
        LanesTexture("lanes_4_stripes_white_14x14m.jpg",4)
    else:                                                # LANE WITH
CENTERLINE: note that centerline is a double line - this influence the mapping
method (pls compare with the non-centerline case below)
        translateUV(0,0,4-NbrOfRightLanes)                #
number of lanes which the texture (with its 8 lanes) must be translated (in v-
dir) so that its center is placed on street center
        LanesTexture("lanes_8_centerline_"+Centerline+"_14x28m.jpg",8)

LanesTexture(texFile,texNLanes) -->
    tileUV(0,~14,0)                                     # the tileUV
operation makes sure that one unit in u-space corresponds to approx 14 meters,
the v-coord is not touched in the case of 0 as parameter
    scaleUV(0,1,1/texNLanes)                            # scaling the v coord for
the texture (e.g. a street with 2 lanes has v coords from 0 to 2, this means it
has to map onto 0 to 2/8 on our texture with its 8 lanes)
    texture(TextureFolder + "Lanes/" + texFile)

Asphalt -->
    tileUV(0,14,14) texture(TextureFolder + "Lanes/asphalt_14x14m.jpg")

# Other default Start Rules for Network Shapes

Joint                --> Lanes
Junction             --> Lanes
Freeway              --> Lanes
FreewayEntry         --> Lanes
Crossing              --> Asphalt
Roundabout           --> Asphalt
RoundaboutIsland     --> Asphalt

# Sidewalk

Sidewalk -->
    split(v,unitSpace,0){ SidewalkHeight: Curbs | ~1: Pavement }

Curbs -->
    extrude(world.y,SidewalkHeight)
    tileUV(0,~2,'1) texture(TextureFolder + "Sidewalks/curbs_2m.jpg")

Pavement -->
    translate(rel,world,0,SidewalkHeight,0)
    tileUV(0,~2,'1) texture(TextureFolder + "Sidewalks/pavement_01_2x2m.jpg")

```

Le fichier portant la texture de l'asphalte est appliqué ici et se trouve dans Assets/Streets/Lanes/asphalt_14x14m.jpg

2) Règle permettant la création et le texturage des toits des bâtiments :

```
/**
 * File:      toit_ardoise.cga
 * Created: 27 Feb 2018 10:08:14 GMT
 * Author:   geoga
 */
```

```
version "2016.1"
```

```
#####3
# Control Attributes
#
```

```
attr FlatRoofTexture    = getFlatRoofTexture
attr SlopedRoofTexture  = getSlopedRoofTexture
```

```
#####
# functions (suited for external usage)
#
```

```
getFlatRoofTexture      = fileRandom("Roofs/Flat/flat*.jpg")
getSlopedRoofTexture     = fileRandom("Roofs/Sloped/ToitArdoise/toit*.jpg")
```

```
#####3
# Helpers
#
```

```
const flatRoofColor = 10%: .6  25%: .7  30%: .8  25%: .9 else: 1
```

```
const slopedRoofTextureDimension = rand(10,15)
```

```
flatRoofTextureDimension(width) =
  case width > 40 : rand(40,80)
  case width > 20 : rand(20,40)
  else           : width
```

```
#####3
# RULES
#
```

```
@StartRule
```

```
Generate -->
```

```
  case geometry.angle(maxSlope) > 2: SlopedPlane
  else                               : FlatPlane
```

```
FlatPlane -->
```

Ces 2 lignes signifient que :

- Si le toit est plat, sa texture sera choisie au hasard parmi les fichiers présents dans Assets/Roofs/flat et donc le nom commence par flat.
- Si le toit est pentu, sa texture sera choisie aléatoirement parmi les fichiers dans le dossier Assets/Roofs/Sloped/ToitArdoise et dont le nom commence par toit.


```

        alignScopeToGeometry(yUp, any, longest)
        setupProjection(0, scope.xz, flatRoofTextureDimension(scope.sx), flatRoofTextureDimension(scope.sz)) projectUV(0)
        texture(FlatRoofTexture)
        color(material.color.r*flatRoofColor, material.color.g*flatRoofColor, material.color.b*flatRoofColor)
        set(material.bumpmap, FlatRoofTexture)

SlopedPlane -->
    setupProjection(0, scope.xy, slopedRoofTextureDimension, slopedRoofTextureDimension, rand(1), rand(1)) projectUV(0)
    texture(SlopedRoofTexture)
    set(material.bumpmap, SlopedRoofTexture)

```

3) Règles permettant la texture des façades des bâtiments :

```

/**
 * File:      Facade_Schematic.cga
 * Created:   10 Apr 2014 19:03:04 GMT
 * Author:    Esri R&D Center Zurich
 */

version "2016.1"

@Hidden
import CIM_Utills:"/ESRI.lib/rules/General/CIM_Utills.cga"

#####
# Control Attributes
#

@Group("Facade Settings",1)
@Order(1)
@Range("Random","Agricultural","Assembly","Educational","Industry","Mercantile","Office","Other","Public","Residential","Service","Transport","Unknown","Utility")
attr Usage = "Random"
@Order(2) @Range(2.7,5.5)
attr UpperfloorHeight = 3
@Order(3) @Range(3.1,7.5)
attr GroundfloorHeight = 3
@Order(4) @Range(2,10)
attr TileWidth = 3.5

@Group("Visualization Options",2)
@Order(1) @Range("grid","vertical","horizontal-open","horizontal-closed","solid")
attr Pattern = _getInitialPattern
@Order(2) @Range("Usage-driven","Blue","Grey")
attr ColorScheme = _getInitialColorScheme
@Order(3) @Color
attr PrimaryColor = _getInitialPrimaryColor
@Order(4) @Color
attr SecondaryColor = "#ffffff"

#####
# Initial attribute settings

```

```

# (to get a diversified default appeareance depending on initial shape,
connection object attributes and randomness)
#

_getInitialPattern =
    case Usage == "Random"      : 35%: "grid" 30%: "horizontal-open" 10%:
"horizontal-closed" else: "vertical"
    case Usage == "Industry"    : "horizontal-closed"
    case Usage == "Office"      : "horizontal-open"
    case Usage == "Residential": "grid"
    else                          : "vertical"

_getInitialColorScheme =
    case Usage=="Random": "Grey"
    else                  : "Usage-driven"

_getInitialPrimaryColor =
    case ColorScheme=="Usage-driven": CIM_Utils.getUsageTypeColor(Usage)
    case ColorScheme=="Blue"        : 20%:"#7090BB" 20%:"#6080AA"
20%:"#80A0CC" 20%:"#7090AA" else:"#90A0BB"
    case ColorScheme=="Grey"        : 20%:"#777777" 20%:"#707070"
20%:"#808080" 20%:"#878787" else:"#909090"
    else                            : "#000000"

#####
#####
###
### RULES
###
###

@StartRule
Generate -->
    alignScopeToGeometry(zUp,any,world.lowest)
    alignScopeToAxes(y)
    FacadeElevationCheck

FacadeElevationCheck -->
    case scope.elevation-initialShape.origin.py > GroundfloorHeight*0.5 #
is face above ground?
        || Pattern == "vertical":
            split(y){ ~1: UpperFloors | UpperfloorHeight*0.5: Wall }
        else:
            split(y){ GroundfloorHeight: GroundFloor | ~1: UpperFloors |
UpperfloorHeight*0.5: Wall }

GroundFloor -->
    case Pattern == "grid":
        split(y){ '0.75: VerticalPattern | ~1: Wall }
    case Pattern == "horizontal-closed":
        split(x){ ~TileWidth: Wall | TileWidth*0.5: Window | ~1: Wall }
    else:
        Window

UpperFloors -->
    case Pattern == "vertical":
        VerticalPattern
    case Pattern == "grid":
        split(y){ ~UpperfloorHeight*0.3: Wall

```

```

        | { UpperfloorHeight*0.5: VerticalPattern
        | ~UpperfloorHeight*0.5: Wall }*
        | UpperfloorHeight*0.5: VerticalPattern }
    case Pattern == "horizontal-open":
        split(y) { ~UpperfloorHeight*0.2: Wall
        | { UpperfloorHeight*0.8: Window
        | ~UpperfloorHeight*0.2: Wall }*
        | UpperfloorHeight*0.8: Window }
    case Pattern == "horizontal-closed":
        split(y) { ~UpperfloorHeight*0.3: Wall
        | { UpperfloorHeight*0.4: Window
        | ~UpperfloorHeight*0.6: Wall }*
        | UpperfloorHeight*0.4: Window }

    else:
        Window

VerticalPattern -->
    split(x) { ~TileWidth*0.6: Wall | { TileWidth*0.7: Window | ~TileWidth*0.3:
Wall }* | TileWidth*0.7: Window | ~TileWidth*0.6: Wall }

Wall -->
    color(SecondaryColor)

Window -->
    set(material.specular.r,0.5) set(material.specular.g,0.5)
set(material.specular.b,0.5)
    set(material.reflectivity,0.3)
    set(material.shininess,20)
    color(PrimaryColor)

/**
 * File: Facade Textures.cga
 * Created: 6 Nov 2013
 * Updated: 10 April 2014
 * Author: Esri R&D Center Zurich
 */

version "2016.1"

#####
# control attributes (set by user)
#

@Group("Facade Settings",1)
@Order(1) @Description("If this attribute is not set (= Random), the rules
randlomy select a texture available for the given height")
@Range("Random","Agricultural","Assembly","Educational","Industry","Mercantile",
"Office","Other","Public","Residential","Service","Transport","Unknown","Utility
")
attr Usage = "Random"
@Order(2) @Range(3,400) @Description("This attribute should be driven by the
master rule. It influences which texture it taken. Note that this value should
be set PER building, and NOT PER facade. Means typically you want the same
texture on a whole building")
attr BuildingHeight = geometry.area/20
@Order(3) @Range(2.7,5.5)
attr UpperfloorHeight = 3.7
@Order(4) @Range(3.1,7.5)
attr GroundfloorHeight = 4.5

```

```

@Order(5) @Range(2,10)
attr TileWidth = 3.5

@Group("Texture Selection",2)
@Order(1)
attr UpperfloorsTexture = getUpperfloorsTexture
@Order(2)
attr GroundfloorTexture = getGroundfloorTexture

# Constants
const LibraryLocation = "Facades/International"
const NeighborLookup = 20 # number of
neighboring textures in the folder to look up to find a texture which fits well
to the building height

#####
# functions (suited for external usage)
#

# attribute dependencies
getGroundfloorTexture = listRandom(_getAllGroundfloorTextures)
getUpperfloorsTexture =
selectGoodUpperfloorsTexture(_getHigherUpperfloorsTextures)

#####
# internal utility functions
#

# file library accessors & utils
_getUsageSearchString = case Usage=="Random": "" else: Usage
_getAllGroundfloorTextures = fileSearch(LibraryLocation +
"/Groundfloors/g_f*" + _getUsageSearchString + ".*")
_getAllUpperfloorTextures = fileSearch(LibraryLocation +
"/Upperfloors/u_f*" + _getUsageSearchString + ".*")
_getHigherUpperfloorsTextures = fileSearch("$" + LibraryLocation +
"/Upperfloors/u_f" + _regexRangeTo999(_calcNbrOfUpperfloors) + "_t[0-9]{1,3}_" +
_getUsageSearchString + ".*")
selectGoodUpperfloorsTexture(list) =
    case listSize(list) > 9 : listRandom(_firstEntries(list))
        # textures with same or higher floorcount found, so we take
the first $NeighborLookup of the list
    else :
listRandom(_lastEntries(_getAllUpperfloorTextures)) # not many textures with
same or higher floorcount found, so we just take the last $NeighborLookup of the
list (i.e. the highest ones)

# file library utils
_calcNbrOfUpperfloors = _round1(BuildingHeight/UpperfloorHeight) - 1
_firstEntries(list) = listRange(list,0,NeighborLookup-1)
_lastEntries(list) = listRange(list,listSize(list)-
NeighborLookup,listSize(list))

# regular expression for height-dependent file search
_hundreds(n) = floor(n/100)
_tens(n) = floor((n/10)%10)
_ones(n) = floor(n%10)

```

Chemin relatif d'enregistrement des textures des façades utilisées par rapport au dossier « Assets ». Ainsi, dans le projet, les textures des façades se trouvent dans : Assets/Facades/International

```

_ above(n) = case n<9: "["+(n+1)+"-9]" else: "X"
# X because otherwise no range needed (and X will have no hits i.e. the
corresponding row in the regex will be useless with X)
_regexRangeTo999(n) =
    "(" + _hundreds(n) + _tens(n) +
    "["+_ones(n)+"-9]" +
    "|" + _hundreds(n) + _above(_tens(n)) + "[0-9]" +
    "|" + _above(_hundreds(n)) + "[0-9]" + "[0-9])"

# reading metadata from filename
_round1(val) = case val>0.5: rint(val) else: 1
_getNbrOfFloors(texture) =
_round1(float(getRange(fileName(texture), "f", "_t")))
_getNbrOfTiles(texture) =
_round1(float(getRange(fileName(texture), "_t", "_")))

# texturing calculations
_randomOffsetU(tex) = rint(rand(0, _getNbrOfTiles(tex)-1)) /
_getNbrOfTiles(tex)
_calcMatchingTextureWidth(tex) =
    case scope.sx > 0.5*TileWidth: scope.sx / _round1(scope.sx/TileWidth) *
_getNbrOfTiles(tex)
    else : scope.sx * _getNbrOfTiles(tex)
* 5
_calcMatchingTextureHeight(tex, floorHeight) =
    case scope.sy > 0.75*floorHeight: scope.sy / _round1(scope.sy/floorHeight)
* _getNbrOfFloors(tex)
    else : scope.sy * _getNbrOfFloors(tex)
* 10

#####
#####
###
### RULES
###
###

@StartRule
Generate -->
    alignScopeToGeometry(zUp, any, world.lowest)
    alignScopeToAxes(y)
    FacadeElevationCheck

FacadeElevationCheck -->
    case scope.elevation-initialShape.origin.py > GroundfloorHeight*0.5: #
is face above ground?
        UpperFloors
    else:
        split(y){ (GroundfloorHeight*_getNbrOfFloors(GroundfloorTexture)):
GroundFloors | ~1: UpperFloors }

GroundFloors -->
    setupProjection(0, scope.xy,
        _calcMatchingTextureWidth(GroundfloorTexture),
        _calcMatchingTextureHeight(GroundfloorTexture, GroundfloorHeight))
    projectUV(0)
    offsetUV(0, _randomOffsetU(GroundfloorTexture), 0)
    texture(GroundfloorTexture)

```

```

UpperFloors -->
    setupProjection(0, scope.xy,
                    _calcMatchingTextureWidth(UpperfloorsTexture),
                    _calcMatchingTextureHeight(UpperfloorsTexture, UpperfloorHeight),
                    0, scope.sy-
    _calcMatchingTextureHeight(UpperfloorsTexture, UpperfloorHeight))
    projectUV(0)
    translateUV(0, _randomOffsetU(UpperfloorsTexture), 0)
    texture(UpperfloorsTexture)

```

4) Règle utilisant les règles de texturage des toits et des façades pour générer l'ensemble du bâtiment :

```

/**
 * File:      Batiment_toit_ardoise.cga
 * Created:   27 Feb 2018 10:08:58 GMT
 * Author:    geoga
 */

version "2016.1"

@Hidden(Usage, BuildingHeight, UpperfloorHeight)
import Facade_Textures:"/ESRI.lib/rules/Facades/Facade_Textures.cga"
(BuildingHeight=Eave_Ht, UpperfloorHeight=Floor_Ht*unitScale, Usage=Usage)
@Hidden(Usage, UpperfloorHeight)
import Facade_Schematic:"/ESRI.lib/rules/Facades/Facade_Schematic.cga"
(UpperfloorHeight=Floor_Ht*unitScale, Usage=Usage)
import toit_ardoise:"/ESRI.lib/rules/Roofs/toit_ardoise.cga"

#####
# Initialize attributes from the attribute table

attr HAUTEUR = 0

#####
# Attributes
#

@Group("Building Settings", 1)

@Order(1) @Range(1, 400) @Description("Distance from ground to bottom of roof")
attr Eave_Ht = HAUTEUR
@Order(2) @Range(1, 400) @Description("Distance from ground to top of roof")
attr Ridge_Ht = _getInitialRidgeHeight
@Order(3)
@Range("Random", "Agricultural", "Assembly", "Educational", "Industry", "Mercantile",
"Office", "Other", "Public", "Residential", "Service", "Transport", "Unknown", "Utility")
attr Usage = _getInitialUsage
@Order(4) @Range("extrusion", "setback top", "setback facade", "setback
base", "setback everywhere")
attr Building_Form = _getInitialBuildingForm
@Order(5) @Range("flat", "shed", "pyramid", "gable", "hip", "half-
hip", "gabled", "gambrel", "mansard", "gambrel-flat", "mansard-
flat", "vault", "dome", "saltbox", "butterfly")
# gable & shed combinations
attr Roof_Form = _getInitialRoofForm
@Order(6) @Range(2.9, 5.2) @Description("in Meters")

```

Cette règle importe les règles précédentes permettant le rendu des toits et façades. Est mentionné le nom de la règle importée ainsi que son chemin relatif dans le projet en cours.

```

attr Floor_Ht          = 3
@Hidden
attr Roof_Ht           = (Ridge_Ht - Eave_Ht) * unitScale

@Group("Visualization Options",2)

@Order(1) @Range("realistic with facade textures","schematic facades","solid
color")
attr Representation = "realistic with facade textures"
@Order(2) @Range(0,1)
attr Transparency    = 0
@Order(3) @Color
attr OverwriteColor = "#ffffff"
@Order(4) @Color
attr RoofColor = OverwriteColor

@Group("Rule Options")

@Order(2) @Range("Meters","Feet") @Description("Unit of Height Attributes")
attr Unit = "Meters"

#####
# Consts
#

# user-driven constants
const unitScale = case Unit=="Feet": 1/0.3048006096012192 else: 1

# for curved roofs such as dome or vault
const curvedAngleResolution = 10

#####
# Functions
#

# for curved roofs such as dome or vault
calcSegmentHt(n) = Roof_Ht * (cos(n*curvedAngleResolution) -
cos((n+1)*curvedAngleResolution))

_getInitialBuildingForm =
    case Eave_Ht*unitScale < 50 : "extrusion"
    case Eave_Ht*unitScale > 100: "setback everywhere"
    else                        : 5%:"extrusion" 15%:"setback top"
15%:"setback facade" 15%:"setback base" else:"setback everywhere"

_getInitialUsage =
    case Eave_Ht>30: "Random" else: 80%:"Residential" else:"Random"

_getInitialEaveHeight =
    case geometry.area < 100 : geometry.area/rand(5,10)
    case geometry.area < 1000: geometry.area/rand(15,25)
    case geometry.area < 7000: geometry.area/rand(10,25)
    else                  : geometry.area/rand(70,200)

_getInitialRidgeHeight =
    case Eave_Ht<30: Eave_Ht+rand(3,6) else: Eave_Ht

_getInitialRoofForm =
    case Ridge_Ht < Eave_Ht+1: "flat"

```

```

    else: 40%: "hip" 50%: "gable" else: "gambrel"

#####
#####
#
# RULES
#
#####
#####

@StartRule
Generate -->
    cleanupGeometry(all,1)
    alignScopeToAxes(y) s('1,0,'1)          # make it horizontal i.e. scale
it flat
    alignScopeToGeometry(yUp, 0, longest)
    set(Eave_Ht,Eave_Ht*unitScale)
    set(Floor_Ht,Floor_Ht*unitScale)
    report("Footprint Area (m2)",geometry.area) report("Nbr of
Floors",rint(Eave_Ht/Floor_Ht))
    set(material.opacity,1-Transparency)
    color(OverwriteColor)
    Footprint

#####
# Building Mass
#

Footprint -->
    case scope.sz < 10 || scope.sx < 10:
        Extrusion(Eave_Ht,true,1)
    case Building_Form == "setback top":
        SetbackTop
    case Building_Form == "setback facade":
        SetbackFacade
    case Building_Form == "setback base":
        SetbackBase
    case Building_Form == "setback everywhere":
        SetbackAll
    else:
        Extrusion(Eave_Ht,true,1)

SetbackTop -->
    split(x){ 'rand(0.1,0.3): Extrusion(Eave_Ht-
rint(rand(3))*Floor_Ht,false,4)
              | ~1                : Extrusion(Eave_Ht,true,6)
              | 'rand(0.1,0.3): Extrusion(Eave_Ht-
rint(rand(3))*Floor_Ht,false,4) }

SetbackFacade -->
    split(z){ 'rand(0.03,0.2): Extrusion(Eave_Ht*rand(0.2,0.8),false,2)
              | ~1                : Extrusion(Eave_Ht,true,6)
              | 'rand(0.03,0.2): Extrusion(Eave_Ht*rand(0.2,0.8),false,2) }

SetbackBase -->
    [ extrude(3*Floor_Ht) Mass(false) ]
    t(0,3*Floor_Ht,0)
    split(x){ 'rand(0.6,0.8): Extrusion(Eave_Ht-3*Floor_Ht,true,6) }

```



```

SetbackAll -->
[ extrude(3*Floor_Ht) Mass(false) ]
t(0,3*Floor_Ht,0)
set(Eave_Ht,Eave_Ht-3*Floor_Ht)
split(x){ 'rand(0.6,0.8):
    split(z){ '0.2: Extrusion(Eave_Ht*rand(0.2,0.8),false,2)
              | ~1 : SetbackTop
              | '0.2: Extrusion(Eave_Ht*rand(0.2,0.8),false,2) }
    }

Extrusion(height,constructRoof,maxLength) -->
    convexify(maxLength)
    comp(f){ all: alignScopeToGeometry(yUp, 0, longest)
ExtrusionConvexified(height,constructRoof,maxLength) }

ExtrusionConvexified(height,constructRoof,maxLength) -->
    case scope.sx < maxLength+1 || scope.sz < maxLength+1: NIL
    else:
        report("Gross Floor Area (m2)",geometry.area*rint(height/Floor_Ht))
        extrude(height) Mass(constructRoof)

Mass(constructRoof) -->
    case constructRoof:
        comp(f){side : Facade | top : Roof }
    else:
        comp(f){side : Facade | top : RoofPlane }

#####
# Roof Generation
#

Roof -->
    case Roof_Form == "shed"           : ShedRoof
    case Roof_Form == "pyramid"        : PyramidRoof
    case Roof_Form == "gable"          : GableRoof
    case Roof_Form == "hip"            : HipRoof
    case Roof_Form == "half-hip"       : HalfHipRoof
    case Roof_Form == "gablet"         : GabletRoof
    case Roof_Form == "gambrel"        : GambrelRoof
    case Roof_Form == "mansard"        : MansardRoof
    case Roof_Form == "gambrel-flat": GambrelFlatRoof
    case Roof_Form == "mansard-flat": MansardFlatRoof
    case Roof_Form == "vault"         : VaultRoof
    case Roof_Form == "dome"          : DomeRoof
    case Roof_Form == "saltbox"       : SaltboxRoof
    case Roof_Form == "butterfly"     : ButterflyRoof
    else                             : FlatRoof

# basic roof types

ShedRoof -->
    roofShed(15) RoofMassScale

GableRoof -->
    roofGable(45,0,0,false,0) RoofMassScale

HipRoof -->
    roofHip(45) RoofMassScale

PyramidRoof -->
    roofPyramid(45) RoofMassScale

```

```

# gable & hip combinations

HalfHipRoof -->
  roofGable(45,0,0,false,0) s('1, Roof_Ht, '1)      # creates a gable roof and
sets its height to the given roof height
  split(y){ '0.5: RoofMass(true)                      # ...
                                comp(f){ bottom: NIL | horizontal:
set(Roof_Ht, Roof_Ht*0.5) HipRoof } } # ... and invokes a hip roof on the top

GabletRoof -->
  roofHip(45) s('1, Roof_Ht, '1)
  split(y){ '0.5: RoofMass(true)
                                comp(f){ bottom: NIL | horizontal:
set(Roof_Ht, Roof_Ht*0.5) GableRoof } }

# gable/hip double-pitched

GambrelRoof -->
  roofGable(70,0,0,false,0)
  split(y){ Roof_Ht*0.7: RoofMass(true)
                                comp(f){ bottom: NIL | horizontal:
set(Roof_Ht, Roof_Ht*0.3) GableRoof } }

MansardRoof -->
  roofHip(70)
  split(y){ Roof_Ht*0.7: RoofMass(true)
                                comp(f){ bottom: NIL | horizontal:
set(Roof_Ht, Roof_Ht*0.3) HipRoof } }

# gable/hip with flat top

GambrelFlatRoof -->
  roofGable(45,0,0,false,0)
  split(y){ Roof_Ht: RoofMass(false) }

MansardFlatRoof -->
  roofHip(45)
  split(y){ Roof_Ht: RoofMass(false) }

# round roofs

VaultRoof -->
  VaultRoof(90/curvedAngleResolution-1)

VaultRoof(n) -->
  case n > 0: roofGable(n*curvedAngleResolution,0,0,false,0)
              split(y){ (calcSegmentHt(n)): RoofMass(n!=1)
                                comp(f){ bottom:
NIL | horizontal: VaultRoof(n-1) } }
  else: NIL

DomeRoof -->
  DomeRoof(90/curvedAngleResolution-1)

DomeRoof(n) -->
  case n > 0: roofHip(n*curvedAngleResolution)
              split(y){ (calcSegmentHt(n)): RoofMass(n!=1)
                                comp(f){ bottom:
NIL | horizontal: DomeRoof(n-1) } }
  else: NIL

# gable & shed combinations

```

```

SaltboxRoof -->
    roofShed(45) s('1,1.5*Roof_Ht,'1)
    split(y){ '0.333: RoofMass(true)
                                comp(f){ bottom: NIL | horizontal:
set(Roof_Ht,Roof_Ht*0.5) roofGable(45,0,0,false,geometry.nVertices-1)
RoofMassScale } }

ButterflyRoof -->
    split(y){ '0.5: roofShed(45,geometry.nVertices/2) RoofMassScale | '0.5:
ShedRoof }

# flat roof

FlatRoof -->
    case Roof_Ht > 0.1:
        RoofPlane offset(-0.4,border) extrude(Roof_Ht) RoofMass(false)
    else:
        RoofPlane

# roof volume

RoofMassScale -->
    s('1,Roof_Ht,'1)
    RoofMass(false)

RoofMass(removeBottomAndTop) -->
    case removeBottomAndTop:
        comp(f){ horizontal: NIL | vertical: Facade | all: RoofPlane }
    else: # remove only the bottom face
        comp(f){ bottom: NIL | vertical: Facade | all: RoofPlane }

#####
# Surface Texturing & Coloring
#

RoofPlane -->
    case Representation == "realistic with facade textures":
        toit ardoise.Generate
    else:
        color(RoofColor)

Facade -->
    case Representation == "realistic with facade textures":
        Facade_Textures.Generate
    case Representation == "schematic facades":
        case OverwriteColor == "#ffffff":
            Facade_Schematic.Generate
        else:
            set(Facade_Schematic.SecondaryColor,OverwriteColor)
            Facade_Schematic.Generate
    else:
        color(OverwriteColor)

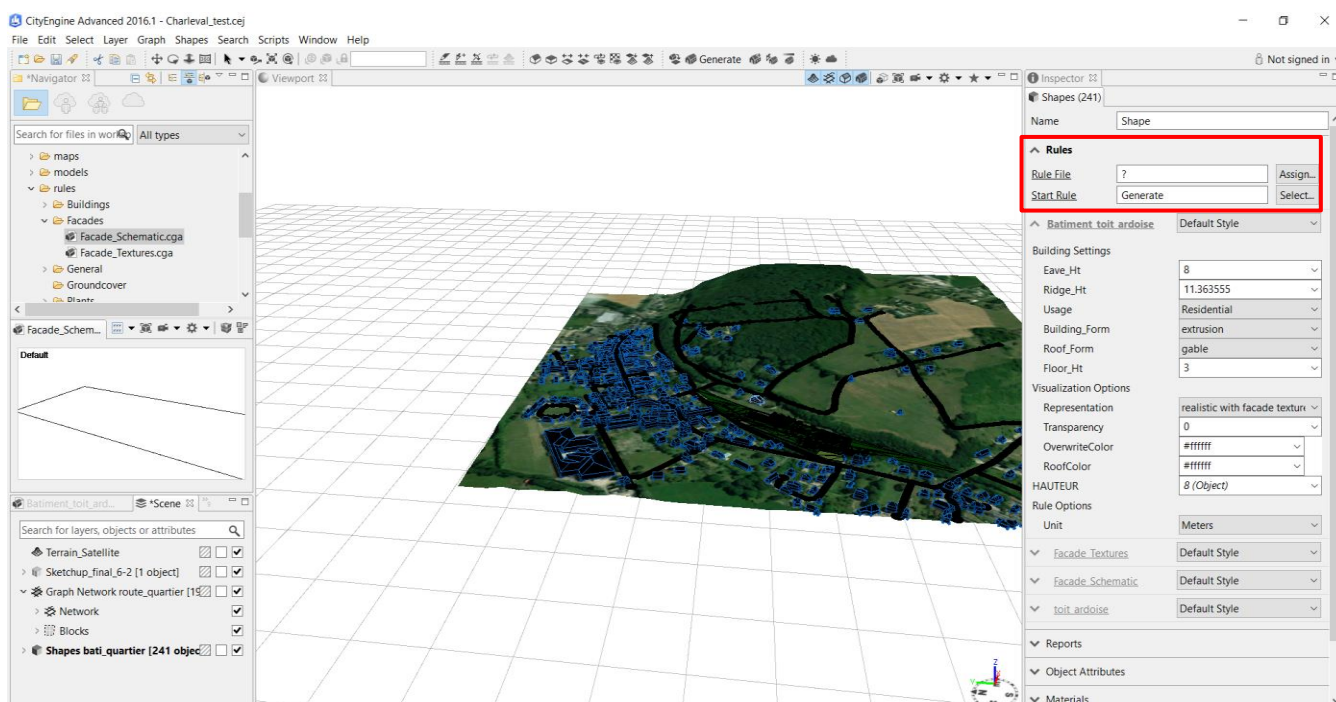
```

Inscrire le nom de la règle générée

Annexe 2 : Protocole de création et d'application de nouvelles textures à partir des règles présentes en annexe 1

Ce protocole permet de créer de nouvelles textures de bâtiments à partir des règles proposées en annexe 1.

- 1) Dans le dossier « Rules » du projet CityEngine, créer de nouvelles règles pour toits, façades (2 règles comme dans l'annexe 1) et bâtiments dans les bons chemins : s'il s'agit d'une règle texturant le toit, dans le dossier Roofs, Clic droit – New – CGA Rule file et la nommer comme souhaité. De même pour les règles texturant les façades dans le dossier Façades et les règles générant l'ensemble des bâtiments dans le dossier « Buildings ».
- 2) Copier et coller les codes de l'annexe 1 relatifs aux toits, façades et bâtiments aux endroits appropriés.
- 3) Trouver les images des textures souhaitées, les dimensionner au format 1024*1024 pixels, et les enregistrer. Il est recommandé de les placer dans le dossier « Assets » du projet CityEngine, dans des dossiers appropriés, par exemple « Roofs » s'il s'agit de textures de toits.
- 4) Dans la règle de texturage des toits (ou des façades), remplacer les chemins relatifs des images précédemment enregistrées.
- 5) Au début de la règle de génération des bâtiments, dans la liste des règles importées, reporter le nom des nouvelles règles utilisées pour les façades et les toits, ainsi que leurs chemins relatifs respectifs.
- 6) A la fin de la règle de génération des bâtiments, reporter également le nom des nouvelles règles dans les lignes encadrées en rouge (nom_règle.generate). Enregistrer toutes les règles nouvellement créées.
- 7) Pour appliquer la règle de génération des bâtiments, sélectionner les bâtiments souhaités, puis, au niveau de l'encadré rouge, sélectionner la règle à appliquer et la ligne de démarrage des instructions (généralement la 1ère ligne dans la fenêtre de dialogue qui s'affiche).



Normalement, si aucune erreur ne subsiste dans le code ou les chemins des fichiers, les nouvelles textures seront correctement générées !

Annexe 3 : Plus de détails sur l'interopérabilité de SketchUp et CityEngine

Dans cette partie, plus de détails sont donnés sur la compatibilité de SketchUp et CityEngine. D'abord, le tableau suivant donne la compatibilité des logiciels avec différents formats de fichier utilisables par des logiciels de nature différente. Les croix indiquent la prise en charge du logiciel pour le format donné.

Format possible	SketchUp	SketchUp Pro	CityEngine
DWG/DXF		X	
Image (PNG, JPEG, GIF,...)	X	X	
STL	X	X	
COLLADA (.dae)	X	X	X
3DS		X	
FBX		X	X
KMZ/KML	X	X	X
OBJ		X	X
VRML (.wrl)		X	
XSI		X	
Script Python			X
ALEMBIC			X
3WS			X
GDB			X
Layer Package			X
Pixar			X
EON			X

*Tableau 4 : Comparatif des compatibilités de SketchUp et CityEngine avec différents formats de fichier
Source : Gaëtan Prabel*

A partir de ce tableau, on peut évaluer l'interopérabilité de SketchUp et CityEngine par rapport à des « groupes de logiciels » permettant des rendus différents. Les résultats sont présentés dans le tableau de la partie 3. Le tableau suivant justifie les cotations proposées :

Type de logiciel	SketchUp	SketchUp Pro	CityEngine
Logiciel de CAO/DAO	Pas compatible	Support du format natif du logiciel AutoCAD (dwg), leader sur ce type de logiciel Note : XX	Support de formats classiques pour ces logiciels tels que collada ou obj Note : X
Logiciel de traitement d'image	Export possible sur des formats informatiques d'image classiques (jpg, png, gif,...) Note : X	Export possible sur des formats informatiques d'image classiques (jpg, png, gif,...) Note : X	Export possible vers format Pixar, pour un rendu haute qualité dans les images 3D Note : XX
Logiciel d'imprimerie 3D	Support du format STL, un des formats permettant l'impression 3D Note : X	Support du format STL, un des formats permettant l'impression 3D Note : X	Pas compatible
Logiciel SIG	Export possible au format kmz, utilisable dans Google Earth ou ArcGIS Earth Note : X	Export possible au format kmz, utilisable dans Google Earth ou ArcGIS Earth Note : X	Export possible au format kmz, et en géodatabase, permettant des analyses supplémentaires dans la suite ArcGIS Desktop Note : XX
Moteur de jeux vidéo	Pas compatible	Export possible au format FBX, utilisable dans les moteurs Unity3D ou Unreal Engine Note : X	Export possible au format FBX, utilisable dans les moteurs Unity3D ou Unreal Engine Note : X

Type de logiciel	SketchUp	SketchUp Pro	CityEngine
Logiciels d'infographie 3D	Pas compatible	Support du format 3DS et OBJ, permettant l'accès à des logiciels comme 3DS Max ou Maya Note : X	Support du format OBJ pour 3DS Max ou Maya, mais aussi de formats plus spéciaux pour des logiciels de rendu de modèle urbain (format EON pour LumenRT) Note : XX
Plateforme Web	Pas compatible	Pas compatible	Support du format 3ws, permettant le visionnage de scènes 3D CityEngine sur le Web, à condition de souscrire au compte ArcGIS Online payant Note : X
Interface de programmation	Pas compatible	Pas compatible	Possibilité d'export sous forme de script Python Note : X

Tableau 5 : Justifications des cotations attribuées pour l'interopérabilité des logiciels SketchUp et CityEngine
Source : Gaétan Prabel

Annexe 4 : Paramètres d'export depuis CityEngine pour les logiciels Unity3D et LumenRT

Voici les paramètres nécessaires au bon fonctionnement du modèle sur des moteurs de jeu vidéo comme Unity3D ou de rendu 3D comme LumenRT. L'exemple pris ici est celui de l'export au format FBX utilisé dans Unity : les réglages sont les mêmes pour le format EON exportable dans LumenRT.

Il est important de créer un nouveau dossier : il contiendra les fichiers FBX mais aussi toutes les textures associées au format image.

Permet d'allouer une taille maximale pour la fichier FBX exporté : ici, CityEngine va exporter un fichier FBX jusqu'à 500 Mo.

Permet de réduire la taille de l'export en groupant les mailles des formes suivant les noms des textures (par exemple asphalte pour les routes).

Ne pas oublier de cocher ces 2 options pour que les matériaux et les textures du modèle soient exportés et affichés...

The screenshot shows the 'Autodesk FBX' export settings window. The 'Current selection' is 241 objects. The 'Preset' is '<Previous Export Settings>'. The settings are organized into several sections:

- General Settings:**
 - Output Path: C:\Années école d'ingénieur\Ecole Ingénieur\PFE\Charleval_quartier_VR (with a 'Browse...' button)
 - Base Name: Charleval_test
 - Export Geometry: Models with Shape Fallback
 - Terrain Layers: Export all terrain layers
 - Simplify Terrain Meshes: ☒
- Granularity Settings:**
 - File Granularity: One file as long as Memory Budget is not exceeded
 - Memory Budget (MBytes): 500
 - Create Shape Groups: ☒
 - Mesh Granularity: Merge meshes by material
- Geometry Settings:**
 - Vertex Normals: Write vertex normals
 - Normals Indexing: Allow shared normals
 - Texture Coordinates: Write all UV layers
 - Local Offset: None
 - Global Offset: ☒ Center (highlighted with a red box and arrow)
 - Vertex Precision: 0.001
 - Normal Precision: 0.001
 - Texture Coordinate Precision: 1.0E-4
 - Merge Vertices Within Precision: ☒
 - Merge Normals Within Precision: ☒
 - Merge Texture Coordinates Within Precision: ☒
 - Triangulate Meshes: ☐
 - Faces With Holes: Triangulate faces with holes
- Material Settings:**
 - Include Materials: ☒
- Texture Settings:**
 - Collect Textures: ☒
 - Create Texture Atlases: ☐
 - Texture Atlas Max Dimension (2^n): 11
 - Texture Atlas Border: ☒
- Advanced Settings:**
 - Write Log: ☒
 - File Type: Binary
 - Embed Textures: ☐
 - Shape Name Delimiter: _
 - Existing Files: Overwrite existing files
 - Script: (empty field with a 'Browse...' button)

At the bottom, there are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

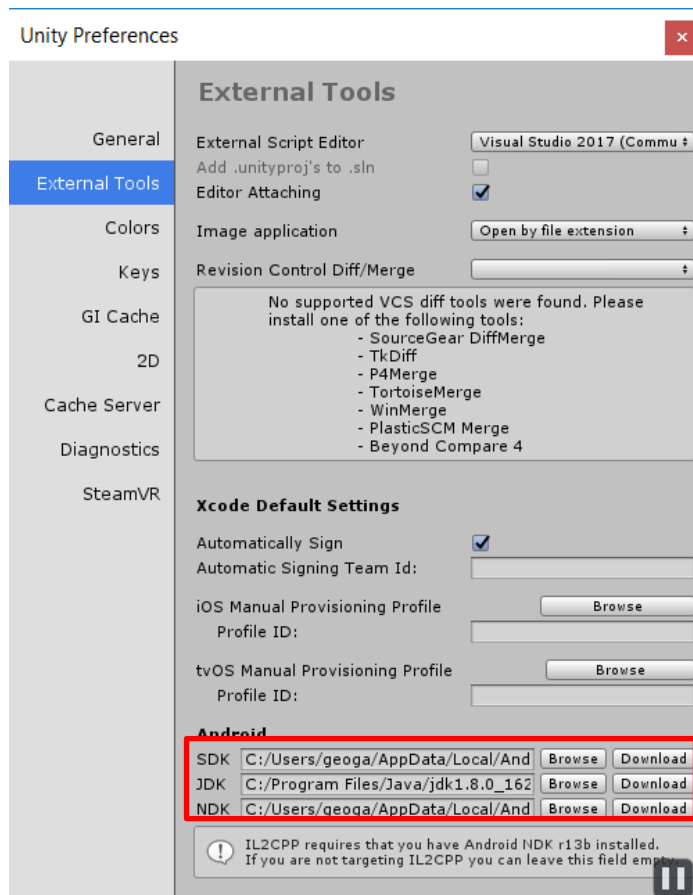
Cette option permet d'exporter tous les terrains, même s'ils ne sont pas sélectionnés.

Cliquer sur « Center » pour centrer l'origine sur le modèle : Unity n'utilise pas de système de coordonnées contrairement à CityEngine, donc il doit placer l'origine sur le modèle et non par rapport au système de coordonnées utilisé.

Annexe 5 : Paramètres nécessaires au fonctionnement d'une application de réalité virtuelle sur Android dans Unity3D

D'abord, lors de l'installation de Unity3D, sélectionner le composant « Android Build Support », pour qu'il soit inclus dans Unity. Dans le projet Unity créé, importer le module « Google VR » se trouvant dans le Tutorial décrivant certains paramètres (voir sources).

Dans Edit – Preferences :



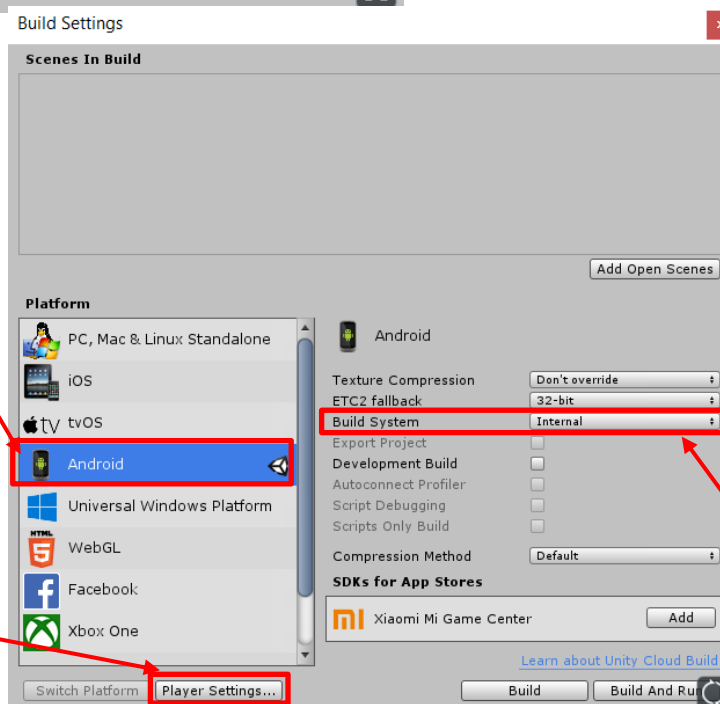
Télécharger et sélectionner les chemins d'accès aux « Development Kit » nécessaires à la construction de l'application :

- Le SDK (Software Development Kit) est à télécharger après l'installation de l'application Android Studio.
- Le JDK (Java Development Kit) : télécharger en version 8.x, les versions 9.x ayant des problèmes de compatibilité avec Unity.
- Le NDK (Native Development Kit).

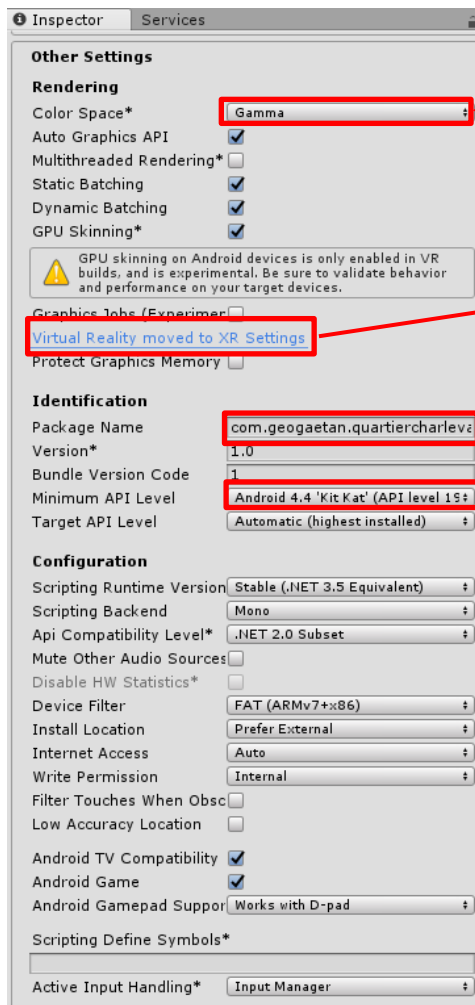
Dans File – Build Settings :

Sélectionner Android comme plateforme de construction d'application avant l'import du modèle CityEngine offre un gain de temps dans l'import.

Régler aussi les « Player Settings », voir page suivante.



Sélectionner « Internal » plutôt que « Gradle » évite des erreurs lors du processus de construction de l'application.



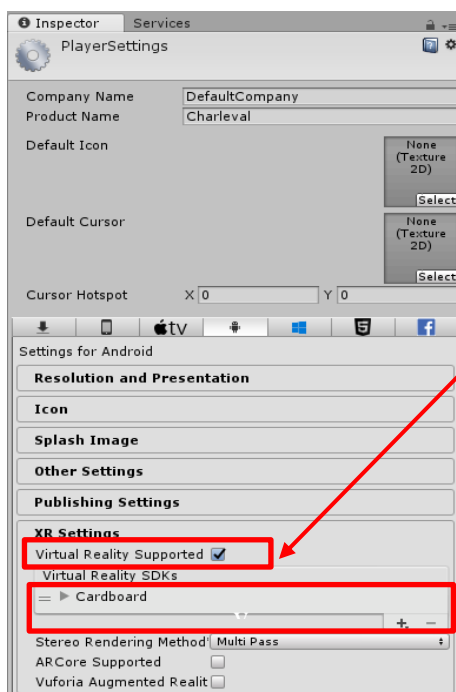
Sélectionner le mode « Gamma », le mode « Linear » n'étant pas compatible avec le système de construction interne, sélectionné au-dessus (« Internal »)

Régler également les « XR settings » pour la réalité virtuelle, voir ci-dessous.

Nommer l'application à construire de la façon suivante :
« com.nom_entreprise.nom_application »
Les lettres et caractères alphanumériques sont autorisés.

Sélectionner la version minimale du système d'exploitation prise en charge :

- 4.4 Kit Kat pour application compatible Google Cardboard
- 7.0 Nougat pour application compatible Daydream
- 5.0 Lollipop pour application compatible Oculus (fonctionne seulement avec les téléphones Samsung à partir du S6).



Activer le support de la réalité virtuelle

Sélectionner la/les plateformes de réalité virtuelle supportées :

- Cardboard fonctionne avec l'application Android Cardboard et les lunettes Google Cardboard.
- Oculus fonctionne avec l'application Android Oculus et le casque Samsung Gear VR.
- Daydream fonctionne avec l'application Android Daydream et le casque Daydream View.

Enfin, dans le smartphone, avant de lancer la construction :

- Installer l'application dédiée à la réalité virtuelle (Cardboard, Oculus ou Daydream)
- Dans les paramètres, activer les options de développement et le débogage USB, et le connecter par USB à l'ordinateur.

35 allée Ferdinand de Lesseps
BP 30553
37205 TOURS cedex 3

Directeur de recherche :
LARRIBE Sébastien

PRABEL Gaétan
Projet de Fin d'Etudes
DA5
2017-2018

Titre : CityEngine : nouveau potentiel de modélisation 3D à l'usage des urbanistes ?

Résumé : Dans le cadre de l'atelier écoconception de DAE4, la modélisation 3D du projet proposé par chacun des groupes est intégrée à l'environnement du quartier sur le logiciel SketchUp. Cette recherche étudie une autre possibilité d'intégration, cette fois grâce au logiciel CityEngine. Un comparatif entre les deux solutions est réalisé : les avantages de CityEngine en matière de prise en charge du Modèle Numérique de Terrain du site, de définition automatisée de textures sur les formes géométriques grâce à des règles, et d'interopérabilité sont détaillés. Les surcoûts engendrés par les deux plateformes sont également étudiés, notamment en matière de prix de licence et d'utilisation des ressources matérielles, pour montrer l'impact financier imposé par une potentielle acquisition de CityEngine. Enfin, ce projet ouvre la voie vers d'autres rendus. D'abord, l'export du modèle depuis CityEngine vers des logiciels de rendu 3D comme LumenRT permet d'aboutir à des rendus plus qualitatifs que ceux actuellement réalisés dans le cadre de l'atelier avec les logiciels SketchUp et Kerkithea. Et puis, un protocole de création d'une application permettant de visionner le modèle en réalité virtuelle sur un smartphone Android a été développé à partir du moteur de jeux vidéo Unity3D.

Mots Clés : Modèle 3D, Modèle Numérique de Terrain, règles de construction de textures, réalité virtuelle.