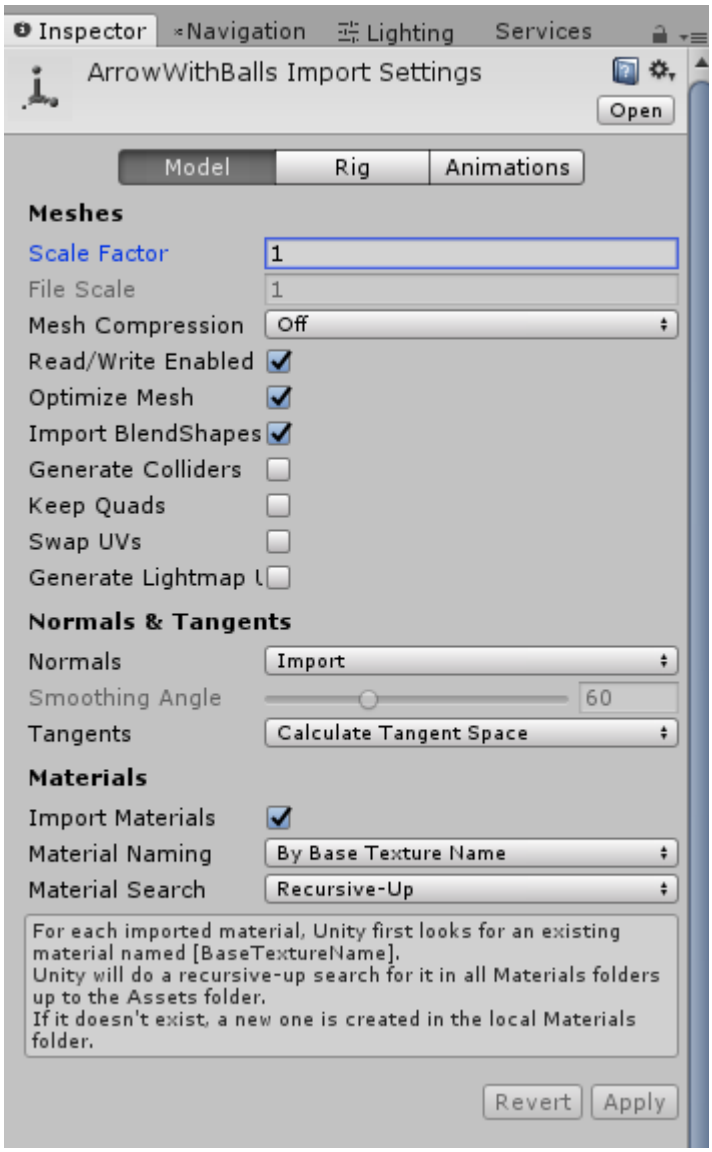
Wstęp, czy coś

W dokumencie punktem odniesienia jest dla mnie Blender. Zgaduję, że Sketchup zachowuje się podobnie.

Import

Nie ma potrzeby eksportować z Blendera do .fbx, wystarczy umieścić plik .blend gdzieś w drzewie katalogów projektu. Każdy plik .blend z punktu widzenia Unity jest czymś na kształt prefabu (Unity nazywa to modelem, nie mylić z meshem, który jest samym kształtem) - możemy z niego korzystać jak z każdego innego prefabu, ale możliwości jego modyfikacji są inne. Kliknięcie pliku da nam w Inspectorze opcje importu.

Każda z opcji ma dość jasny tooltip i jest raczej intuicyjna. Niektóre jednak opiszę dokładniej:



- Import BlendShapes: dotyczy techonologii używanej jedynie w Mayi, a służy bodaj do animowania twarzy. Dla nas bez większego znaczenia.

- Generate Colliders: włączenie sprawi, że obiekt (obiekty), które powstają po umieszczeniu modelu będzie miał mesh collider oparty o odpowiedni mesh. Raczej bezużyteczne, gdyż mesh colliderów staramy się używać rzadko, poza tym i tak zwykle ręcznie je ustawiamy (np. obiektowi, który ma skomplikowany mesh dając mesh collider oparty o mesh dużo prostszy).

- Keep Quads: w teorii ma sprawić, że prostokaty nie będą krojone na trójkąty. Próbowałem się bawić, psuło mi jedynie tekstury.

- Swap UVs: jeśli obiekty mają lightmapy i zamiast tekstur mamy lightmapy (i na odwrót), to to może pomóc (nie znam się na lightmapach, nigdy nie miałem takiego problemu).

- Generate Lightmap UVs: jeśli dobrze rozumiem, zaznaczamy jeśli chcemy na tym obiekcie bejkować lightmapy.

- Import Materials: Jeśli model jest gotowy i kompletny, z materiałami włącznie, to chcemy importować materiały. Osobiście preferuję tworzenie materiałów w samym Unity, stąd ta opcja nie jest dla mnie atrakcyjna.

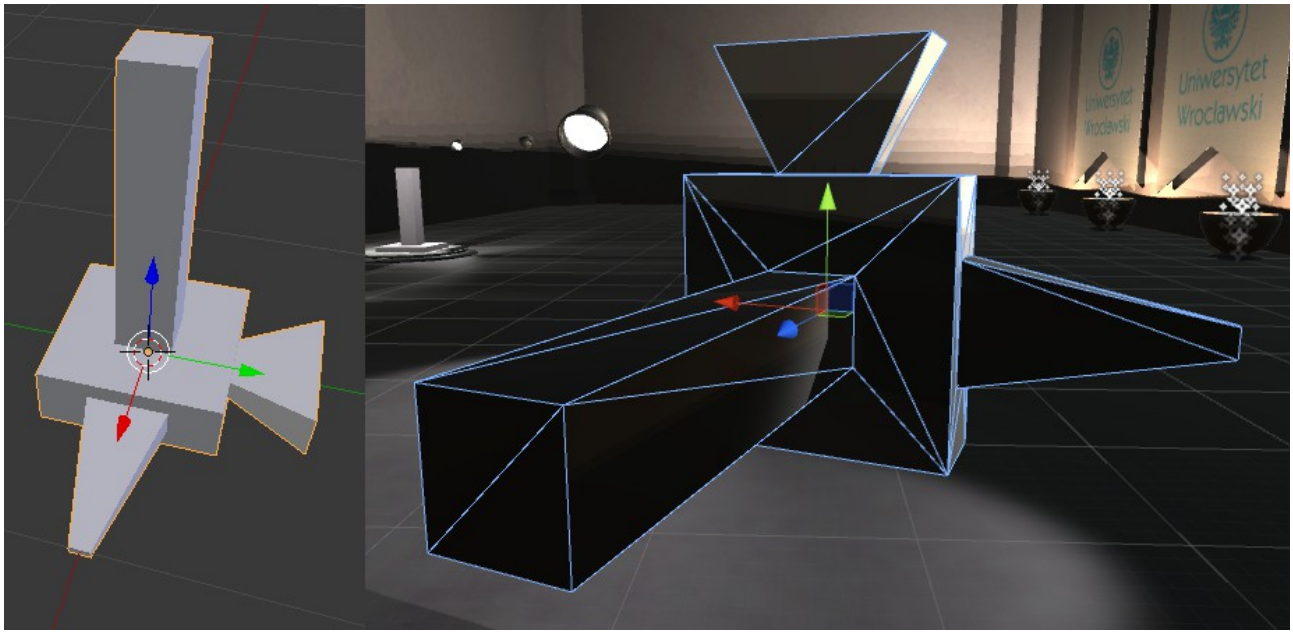
Zakładki Rig oraz Animations dotyczą kolejno szkieletu oraz animacji.

Podobnie można zmienić ustawienia importu tekstur - kliknięcie na plik graficzny da nam w inspectorze ustawienia. Są dość intuicyjne, jednak na specjalną uwagę zasługuje Texture Type "Advanced", który pozwala nam m. in. wygenerować normalmapę z czarno-białej heightmapy. Poza tym, najważniejsze ustawienia to typ (normal map, texture etc) i wrap mode, który określa, jak traktować vertexy, których współrzędne leżą poza przedziałem (0,1). W przypadku "wrap" dostajemy część ułamkową ze współrzędnej (zawija się), w przypadku clamp wszystkie te punkty mapują się na najbliższy, którego współrzędne leżą w tym przedziale.

Ważnym jest, że nie musimy umieszczać na scenie całego modelu. Każdy plik z modelem można rozwinąć i dostać się do jego poszczególnych składników - konkretnych meshy, animacji i tak dalej.

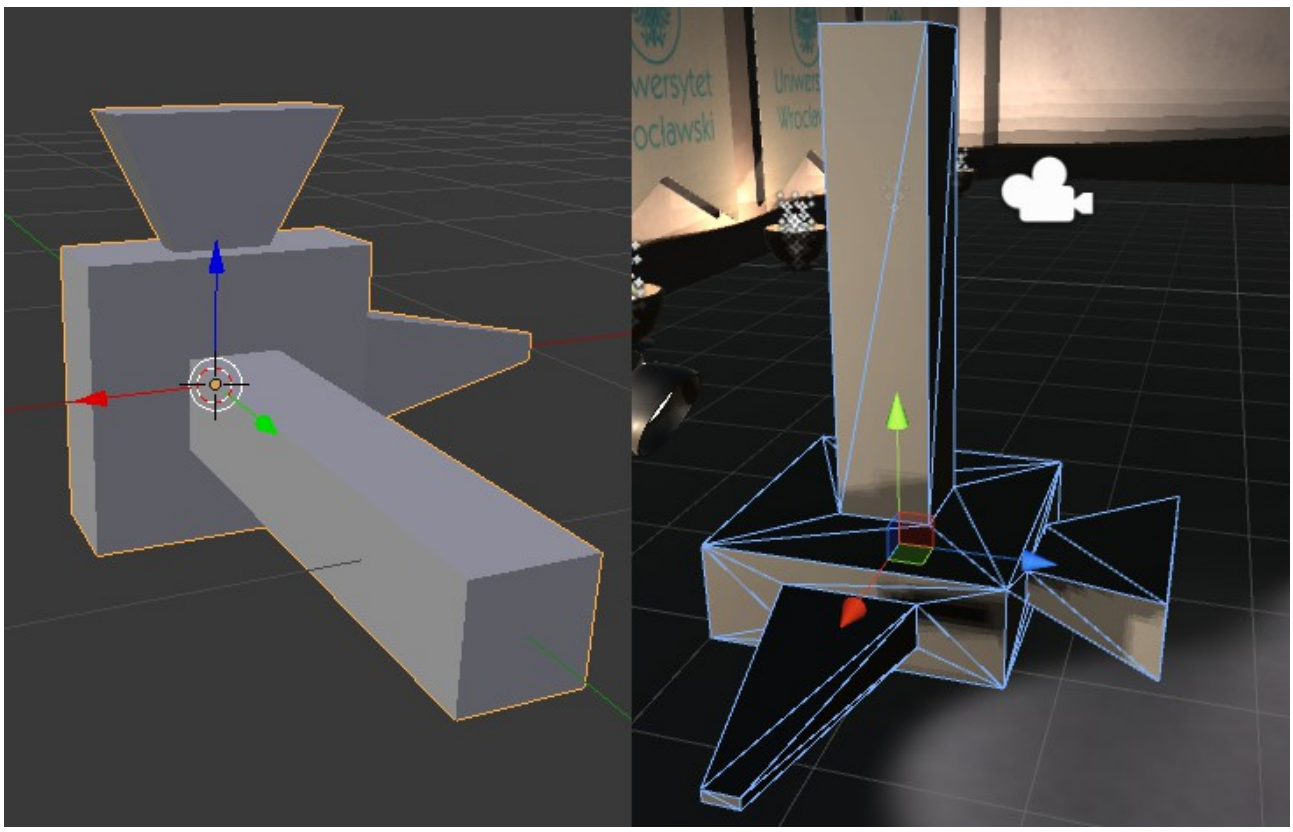
Osie Blendera, a osie Unity

Obiekt o rotacji zerowej będzie w Blenderze leżał inaczej, niż w Unity. Oś Z oraz Y będą się zgadzać z oryginałem, zaś X będzie odwrócona. Sprawę komplikuje fakt, że w Unity role osi Z i Y są zamienione w stosunku do Blendera.



Po lewej: obiekt w Blenderze. Po prawej: to, jak wygląda po imporcie w Unity.

Wstawiając gotowy prefab powstały w skutek Unity dodaje mu rotację, by jego lokalna oś Z pokrywała się z globalną osią Y (-90 stopni wg. osi X), a następnie dopiero aplikuje rotacje takie, jak posiadał obiekt oryginalnie. W większości przypadków wystarcza to, by obiekt umieszczony na scenie wyglądał tak, jak sobie życzymy. Jeśli jednak zależy nam na odnoszeniu się np. do **`transform.forward`** obiektu, to dobrze jest odpowiednio przygotować obiekt do eksportu - tak, aby był zwrócony górą wg. osi Y, przodem wg. osi Z i prawą stroną wg. odwróconej osi X. Jeśli oryginalnie góra, przód i prawo były osiami kolejno Z, Y i X, oznacza to rotację o -90 stopni wg. X i 180 wg. Y. Należy pamiętać, by obroty te zaaplikować do modelu. Warto też dodać (ale nie aplikować) obiektowi rotację o 90 stopni wg. osi X, by znosiła się z tą, którą dodaje Unity.



Po lewej: odpowiednio spreparowany obiekt w Blenderze. Po prawej: pożądany efekt uzyskany w Unity.

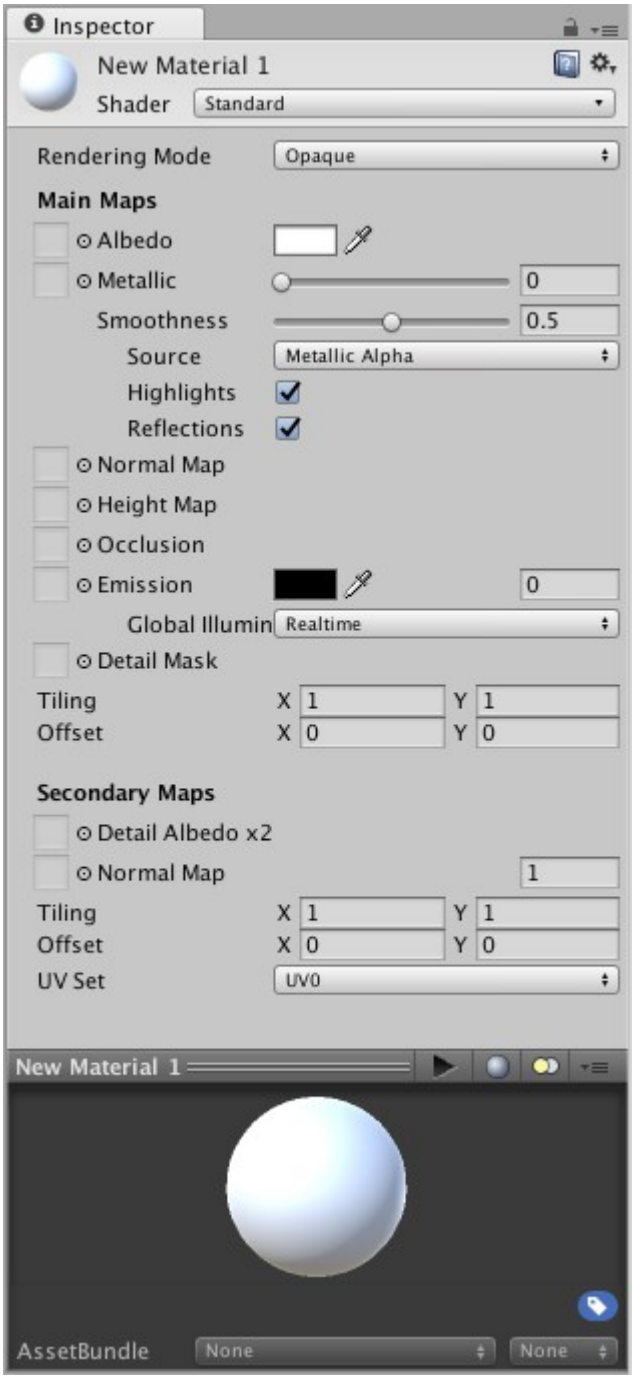
Skala

To zarówno kwestia dobrej praktyki, jak i instynktu samozachowawczego: w Unity chcemy mieć jak najmniej obiektów zeskalowanych, zwłaszcza jeśli chodzi o skalę niejednorodną. Zachowanie dziecka obiektu zeskalowanego może być inne, niż byśmy oczekiwali. Dlatego tworząc modele/skomplikowane prefaby/cokolwiek, starajmy się, by obiekty, które nie są liśćmi miały skalę (1,1,1) - co w przypadku przygotowywania modelów oznacza tak naprawdę wszystkie, gdyż często nie wiemy, czy będziemy chcieli pod modelowany obiekt podpiąć w grze jakieś dziecko.

Wiele obiektów w jednym pliku

Bardzo często obiekt składa się z więcej, niż jednego mesha. Unity szanuje strukturę dziecko-rodzic, którą ustaliliśmy w Blenderze i wiernie ją oddaje. Jeśli jest więcej niż jeden obiekt, który w Blenderze nie miał żadnego rodzica, Unity utworzy pusty GameObject, którego dziećmi będą te wszystkie obiekty. Tylko one otrzymają wspomnianą wcześniej rotację o -90 stopni wg. osi X. Dodatkowo, rozwijając plik dostajemy do nich dostęp (co umożliwia umieszczenie tylko jednego z nich, co przydaje się, jeśli obiekty te są niezależne od siebie, ale chcemy mieć je w jednym pliku).

Tekstury i materiały



To, jak tekstura leży na obiekcie zależy od jego UV layoutu, który określa, jak kolejne vertexy mesha mapują się na płaską teksturę. Można też fragmentom mesha przypisać różne materiały - to wszystko robimy w programie do modelowania.

W Unity możemy materiały zaimportować, lub nałożyć własne. Opiszę proces tworzenia materiału opartego o Standard Shader przez wymienienie, za co odpowiedzialne są poszczególne opcje.

- Rendering mode:

- Opaque: zero przezroczystości.

- Cutout: nie ma fragmentów półprzezroczystych, kanał alfa jest zaokrąglany do 0 lub 255.

- Transparent: Pozwala na częściową przezroczystość, dobre do materiałów takich jak np. szkło.

- Fade: Pozwala na częściową przezroczystość, działa nieco inaczej niż transparent, polecają używać do rzeczy częściowo zanikłych (np. hologramy).

- Albedo: źródło koloru materiału, umieszczamy tu podstawową teksturę, można ją zabarwić (gdzie białe zabarwienie to oryginalny kolor). Brak tekstury daje kolor biały.

- Metallic: najlepiej poeksperymentować, bo żadne wyjaśnienie nie przedstawiło mi tego lepiej, niż wiele prób i błędów. W najprostszym scenariuszu ustawiamy po prostu wartości, jednak jeśli materiał zawiera np. elementy skórzane i metalowe potrzebujemy specjalnej tekstury. Kanał czerwony określa w niej poziom metaliczności, alfa gładkość (stąd miejsca metalowe będą na niej na pewno bardziej czerwone).

- Normal Map: tu można umieścić mapę normalnych, która pozwala nadać efekt trójwymiarowości płaskiej powierzchni. Częstą praktyką jest generowanie jej z bardziej

skomplikowanego mesha w programie do modelowania, lub wykorzystanie czarno-białej heightmapy (ciemniej - wgłębienie, jaśniej - wybrzuszenie), którą możemy zaimportować jako normal mapę.

- Height Map: wykorzystuje czarno-białą teksturę i daje efekt podobny do Normal Mapy, z tym że skutkuje faktycznym odkształceniem widoku, zamiast grą światłem i cieniem.

- Occlusion: mówi, które miejsca powinny dostawać mniej, a które więcej światła niebezpośredniego. By osiągnąć jak najlepszy realizm chcemy, by zagłębienia (np. fałdy ubrań) dostawały mniej światła.

- Emission: określa własne światło obiektu pokrytego tym materiałem w odróżnieniu do Albedo, które określa światło odbite. Przykładowo włączony ekran komputera powinien być dobrze widoczny nawet nieoświetlony.

- Detail mask: jeśli używamy detali, pozwala nam kontrolować, gdzie i w jakim stopniu powinny być nałożone

- Tiling i offset: kontrolują, jak tłumaczyć współrzędne pikseli tekstur na współrzędne UV vertexów.

- Detail Albedo i Normal Map: pozwala nałożyć dodatkowe albedo i mapę normalnych na te oryginalne, np. celem dodania porowatości. Ze względu na niezależny tiling i offset mogą być małe i odpowiednio zagęszczone (przez zwiększenie tilingu).