절차적 콘텐츠 생성: 모듈로 생각하기

(Procedural Content Generation: Thinking With Modules)

작성자: 이치로 람베(Ichiro Lambe)

작성일: 2012 년 7 월 5 일

이상적인 절차적 콘텐츠 생성(Procedural Content Generation) 알고리즘이 있다면 우리는 버튼 하나만 눌러서 세계를 창조할 수 있다.

"세계 창조"

물론 이것이 상당히 어려운 문제라는 것이 드러났으므로, 아직 그 단계에 이르렀다고 말하기는 어렵다. 하지만 이 글을 통해 그 단계에 한 걸음 더 가까이 다가가기 위한 몇 가지 고찰을 전달하고자 한다.

절차적 콘텐츠 생성(Procedural Content Generation, 이하 PCG), 즉 배경을 이루는 장면에서부터 스토리 라인의 조화까지 모든 것을 알고리즘적으로 만들어낸다는 아이디어는 너무나 매력적이다. 그렇지 않은가?

수작업으로 게임 환경을 만들어내는 것은 시간이 걸리고, 이것을 모두 저장하는 데에도 많은 공간이 필요하다. <Starflight>와 <Elite> 이후로, 개발자들은 제한 없이 창의성을 발휘할 수 있는 지점을 향해 작업을 계속하고 있다.

크게 보아 개발자들은 주로 다음의 세 가지 이유에서 PCG 를 사용한다:

- 개발자로 하여금 더 빠르게 콘텐츠를 생성할 수 있도록 해준다.
- 게임이 플레이어의 동작에 실시간으로 반응할 수 있도록 해준다.
- 콘텐츠가 디스크에서 차지하는 용량을 줄여준다.

그리고 숨어있는 또 다른 이유도 있다:

- 실험을 통해 더욱 창의적인 생각을 할 수 있게 해준다.

이 글에서, 우리는 2009 년에 우리가 만든 게임 <AaaaaAAaaaAAAaaAAAAaAAAAA!!! -- A Reckless Disregard for Gravity (줄여서 Aaaaa!)> 와 아직 출시 예정인 <1... 2... 3... KICK IT! -- Drop That Beat Like an Ugly Baby (줄여서 Ugly Baby)>를 만들며 찾아낸 PCG 의 역사와 문제, 해결책과 그 방법에 대해 논할 것이다.

절차적으로 생성한 레벨
초특가 할인
레벨 디자이너는 이제 가라
한정판: 무료 브릿지 포함

스포일러: 우리는 PCG 의 단점을 보완하고 이점을 부각시키는 모듈식, 그래프 기반의 시스템을 선호한다. 한 줄 요약을 하자면 그렇다. 이제 남은 글을 읽던가, 알라스카 썰매개 조련사가 되던가 하는 건 모두 당신 마음이다.

게임에서의 성공

첫째로, PCG 가 때로는 - 늘 눈앞에 있지만, (아직) 실용화되지 않았단 점에서 – 하늘을 나는 자동차처럼 보일지라도, 실제로 실행가능하며 유용하다는 증거가 있다.

<로그(Rogue!)> 는 여전히 절차적 콘텐츠 생성을 게임에 적용한 훌륭한 예로 꼽힌다. 1980 년 즈음에 만들어진 이 게임은 컴퓨터 자체에 권한을 부여하여 우리가 플레이하는

판타지 세계를 만들어 내게 했다. 이를 통해 지하 세계와 미로처럼 꼬인 길을 구축하고, (미리 창조된) 물약이나 적, 무기 등으로 채워놓는다. 이런 스타일의 던전 생성은 (Hack, Moria, Larn, Nethack, Angband, Dungeon Siege, Dungeon Siege Ⅱ, Diablo, Diablo Ⅱ, Diablo Ⅲ 외 다수에서 볼 수 있듯이) 성공적이었고, 많은 개발자들이 로그라이크(roguelike) 게임을 만들었고, 많은 자원이 로그류 개발에 투입되었다는 점을 볼 때 비교적 잘 연구된 편이다.

로그(Rogue) 없이 PCG 를 논한다는 것은 불가능한 일이지만, 아타리 ST 용 Temple of Apshai Trilogy 의 스크린 샷을 사용하였다. 별 이유는 없다.

<Starflight> (1986)와 그 속편의 어마어마한 판매량에 힘입어, 엄청난 양의 행성 혹은 세계 탐험에 관한 게임이 쏟아져 나왔다. 각각의 태양계 안에는 많은 양의 행성이 포함돼 있었고, 또 각 행성들은 서로 다른 특성(표면 온도, 중력, 날씨, 대기, 습도)을 지니고 있었다.

그 시절 게임이 특히 놀라웠던 점은, 행성계에 착륙하면, 구불구불한 해안선과 산을 탐험하며 행성 타임과 고도에 따라 다양한 밀도를 갖는 광맥(알루미늄, 몰리브덴, 그 외 다수)과 생명체(고정형과 이동형)를 발견할 수 있었는데, 그런 행성계의 수에 제약이 없었다는 점이다. 원작은 양면 5.25 인치 플로피 디스크에 다 들어가는 용량이었다. Braben/Bell 의 고전 게임 <Elite> (1984)는 어쩌면 플레이어가 날아가서 트레이드를 할 수 있는 행성의 수가 은하계 8 개와 맞먹는 게임으로 더 잘 알려져 있을 지도 모른다.

Starflight 2 에는 심지어 플레이어가 거래를 할 수 있는 마을까지 포함돼 있다.

그보다 최근의 예로, <스포어(Spore)>는 절차적 모델 생성과 애니메이션을 보여주고 있다. 이 게임에서 플레이어는 크리쳐(creature)를 만들 때 뼈의 길이와 두께를 마음대로 정할 수 있으며, 팔다리나 눈, 귀, 날개 등을 더할 수도 있다. 창의성을 게임 플레이의 한 부분으로 만든 것이다.

스포어에서의 크리쳐 생성 튜토리얼(video: http://www.youtube.com/watch?v=ZRr3lgckIAM)

그리고, 몇 년 전 <.kkrieger>는 1 인칭 슈팅 게임 전체를 이 글의 용량보다 작은 디스크 공간에 집어 넣어서 온 세상을 놀라게 했다

단 97,280 바이트를 사용해 만든 <.kkrieger>

PCG 는 게임의 역사를 관통하여 사용되어왔고, 오늘날에도 여전히 사용되고 있다. 당연히 모든 게임에 사용되어야만....

....하는가?

제작 도구로서의 PCG

우리의 2009 년작 <Aaaaa!>에서 우리는 자동화된 도구와 창의성을 목표로 한 도구 두 가지 모두를 사용하고 싶었다.

<Aaaaa!>는 PCG 로 질감과 레벨을 생성한 후, 수작업으로 수정하였다.

이미지를 한 픽셀씩 칠해본 적이 있는가? 혹은 메모리 위치에 값을 하나하나 지정해본 적은? 너무나 지루한 일이기 때문에 개발자들은 디지털 아티스트를 위해 더 좋은 툴을 개발해 냈다. 그래서 요즘은 마우스를 움직이는 것으로 픽셀을 칠하고, 색칠이 된 사각형을 그리거나 그래디언트 텍스쳐를 렌더링할 수도 있다.

자동화는 중요한 것이다. 시간을 절약해주고, 우리가 원하는 것을 대체로 정확히 만들어준다. 캔버스 위의 한 점을 클릭하고, 다른 한 점을 클릭한다. 이것으로 정확히 우리가 원했던 대로 칠해진 사각형을 얻을 수 있다.

<Aaaaa!>는 미래 메사추세스주 보스턴의 공중 빌딩 숲을 배경으로 한 베이스 점핑 게임이다. 우리는 게임 내 편집기를 이용해 빌딩과 교각, 도보와 신호체계, 비행 자동차와 거대 감자와 같은 콘텐츠를 하나하나 수작업으로 만들어 냈다.

이 툴은 기술적으론 나무랄 데 없었다. 하지만 어느새 우리가 매번 비슷한 것만 계속해서 만들어내고 있다는 사실을 깨달았다. 이 툴이 작업의 어떤 특정 부분은 쉽게 만들어 주는 반면 다른 작업은 여전히 어려운 것이 한 가지 이유였다.

예를 들어, 레벨 편집기 상에서 건물 몇 개를 세우고, 이것을 점수판으로 꾸미는 일은 쉬웠다:

그러나 더 복잡한 뭔가를 만들어내는 일은 손이 많이 가는 지루한 일이었고, 우리가 넌덜머리를 내며 이것을 하나하나 손으로 해 나가는 동안 (아마 이것 땜에 레벨 디자이너를 더 많이 채용했을 것이다) 자동화라는 해결책이 등장했다. 예를 들어 스크립트가 점수판을 일렬로 생성하면, 그 다음에 우리가 패턴을 만드는 것이다.

자연스레 다음 단계는 무작위로 배열된 사인(sin) 곡선에 따라 판들을 흩어놓는 것이었다.

```
#사인 곡선에 따라 점수판 40 개 만들기

    for i in 0..40:

        plate.x = sin(i*freq1)*amplitude

        plate.y = sin(i*freq2)* amplitude
```

진짜 재미난 일은 우리가 뭔가 웃기는 것을 만들기 위해 거기에 고주파를 쏘기 시작했을 때 일어났다:

플레이가 전혀 불가능한 것들이 튀어나왔지만, 그랬기 때문에 **진짜 재미있었다**. 레벨 디자인에 매몰되어 있던 우리에게 이런 것들이 전혀 예측하지 못한 새로운 방식의 재미를 주었다.

이것은 "절차적으로 생성한" 레벨의 사소하고 간단한 예이지만, 이렇게 간단한 스크립트를 사용할 때에도 우리를 웃게 만드는 상황을 적어도 한 번 이상 만나곤 했다. 이것은 우리가 레벨을 구축하는 방식을 바꾸어 놓았고, 플레이어들에게는 새로운 도전을 제시하였다.

곧 우리는 "레벨 뼈대(level skeletons)"라고 부르는 유용한 스크립트의 작은 콜렉션을 갖게 되었다. 우리를 즐겁게 만드는 것들을 만들어내기 위해 스크립트들을 마구 사용했고, ("아! 이건 의도하지는 않았는데, 근사하잖아!"), 그 다음에 나머지 레벨을 수작업으로 만들어 냈다.

<Aaaaa!>는 정말 잘 풀려서 다른 상 뿐만 아니라 IGF 의 후보에 오르기까지 했다. 여기까지 우리가 얻은 자신감에 들떠서, 큰 맘 먹고 말해 보기로 했다. 우리는 다음 게임에서 PCG 를 레벨 디자인 전체에 사용할 것이다. 사실 우리는 그래도 꽤 괜찮은 프로그래머들인데... 그냥 우리가 전부 프로그래밍하는게 더 쉽지 않았을까?

(스포일러: 그렇지 않았다.)

PCG 는 암도 고친다

우리의 차기작 <Ugly baby>는 <Aaaaa!>와 비슷한 방식의 게임이기는 하다. 하지만 우리는 이 게임의 전체 레벨 구조를 알고리즘적으로, 런타임(runtime)에 플레이어가 고른 미디어를 기반으로 생성해내고 싶었다. 이러한 미디어는 Audiosurf 나 Beat Hazard 와 같은 음악에서부터 미국 독립 선언을 찍은 비디오에 이르기까지 모든 것이 될 수 있다.

우리는 이 게임을 이렇게 묘사한다:

"당신이 제일 좋아하는 댄스곡에 맞춰서 게임을 즐기세요. 혹은 트랜스 음악을 들으며 둥실 떠오르는 기분을 만끽하세요. 'Ugly Baby'는 당신의 MP3 음악으로 당신이 싸워나가야 하는, 공중 세계를 창조해 드립니다."

Video: http://www.youtube.com/watch?v=A3W-r9IobwI

우리가 PCG 를 통해 바랬던 것은 다음과 같다. (그리고 지금도 바라고 있다).

- <Aaaaa!> 에서 수작업으로 만든 것보다 더 뚜렷하고 재미있는 레벨을 무한대로 만들어 낼 것

- 모든 레벨을 플레이어가 가진 미디어에 기반하여 런타임에 바로 생성해 낼 것
- 플레이어가 "레벨 DNA"와 자신의 레벨을 직접 "성장"시키면서 게임 세계를 창조해 내는 데 참여할 수 있도록 할 것

우리는 음악을 읽어들여 <Aaaaa!>와 같은 레벨을 만들어내는 스크립트를 만들고 싶었다. 공중 구조물에 대한 우리의 이해를 구현하고, 대상을 단순화하여 표현하는 제너러티브 아트(generative art)[1]를 이용해 적을 만들어 냄으로써 말이다. 9 개월짜리였던 프로젝트는 우리가 본격적으로 이를 시작하기도 전에 이미 24 개월짜리가 되어버렸으며, 우리는 4 가지의 커다란 문제에 봉착하게 되었다:

1. 수작업의 강점은 PCG 의 약점으로 작용했다.

알고리즘을 사용하는 것과 수작업으로 콘텐츠를 생성해내는 것은 대체로 상호 보완관계에 있다고 할 수 있다. 거대한 산이 있고, 그 산이 침식된 모습을 보고 싶다면, 알고리즘이 수작업보다 훨씬 유용하다. 손으로 일일이 다듬는데 온 하루를 써야 할 때, 침식 툴은 몇 초 안에 이를 해결해 주고 다른 것을 할 수 있게끔 해준다.

이와 반대로, 만약 동굴 입구에 나무 몇 그루를 추가한다고 할 때는, 스크립트를 짜는 것보다 오히려 수작업으로 나무를 심고 살살 손을 보는 것이 더 낫다. 해변에 "HELP"라는 글자를 새겨 넣을 때도, 툴을 골라서 글자를 그려 넣는 것이 더 쉽다. 우리는 이것을 다음과 같은 삽질을 통해 깨닫게 되었다.

- 1 단계: 레벨 골격을 생성하는 알고리즘 짜기 (1 시간)
- 2 단계: 빌딩이 너무 멀리 떨어져 있지 않나 살펴봐가면서 밀도를 높이되 서로 겹치지는 않게 주의할 것. (15 분)

[1] 자동 생성, 우연 생성에 의한 작품을 만들어내는 미술 장르 (역주)

- 3 단계: 왜 전혀 안 되나 테스트해보면서, 빌딩들이 좀 더 비껴서 서 있도록 길을 기울여 보기.(15 분)
- 4-9 단계: 지우고, 새로 해 보기 (한 단계에 15 분씩)
- 10 단계: 손으로 하면 쉬운 걸 이런 식의 접근이 통하지 않았음에 욕을 해대기 (5 분)

이런 과정을 통해 우리는 기대해볼 만한 스크립트를 만들어 냈다고 믿기 시작했고, 끊임없는 수정을 통하여 일단은 맥시멈이라고 할 만한 결론에 도달하였다. 초반의 스크립트로 레벨 레이아웃의 특정한 유형을 만들어 내고, 다른 것을 시도하기보다는 만들어진 유형으로 최적의 순열을 찾아내는 것이다. 우리는 후에 "재미 없음"과 "재미 있음"이 종이 한 장 차이라는 것을 알게 되었다. 하지만 알고리즘이 됐든 재미 요소를 보강하는 것이든 실행하는 건 시간이 걸리는 일이다.

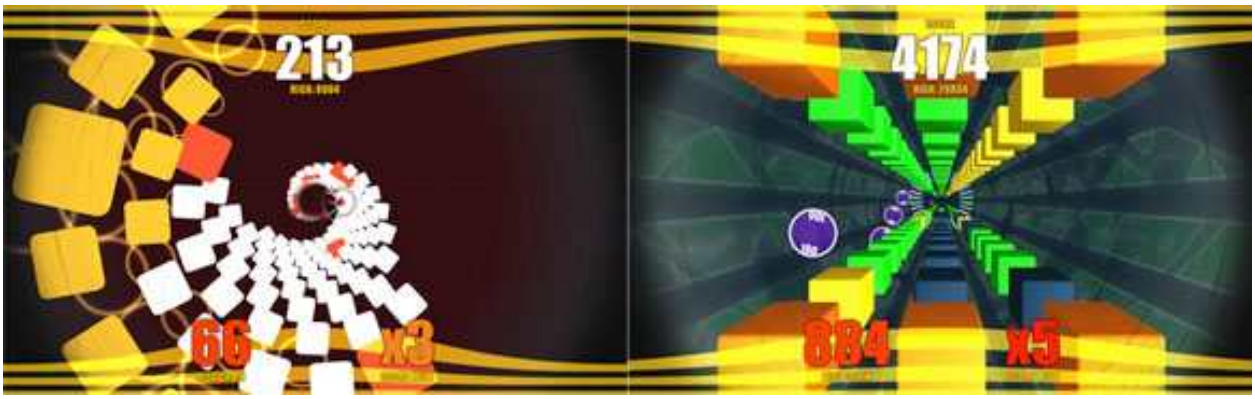
해결책: 때로는 손으로 시험해 보는 것이 낫다. 만들어보면서 학습하고, 그것을 흉내내는 알고리즘을 만들어 좋은 결과가 나오게 만든다.

2. 반복적인 콘텐츠를 만드는 것은 쉽다.

여기서 Alex Norton 의 AltDevBlogADay post 를 인용해야겠다:

> 왜 애초에 한계를 정하는가? 절차적 코드 생성은 지난 25 년간 그다지 많이 변화하지 않았다. 사람들은 아직도 모든 것에 줄기차게 프랙탈(fractal)과 다이아몬드, 물방울을 사용한다. 너무 우려먹었을 뿐 아니라, 그저 절차적으로 콘텐츠를 생성하는 것처럼 보이기만 하는 것들 말이다. 모든 프로그래머에게 이것은 단지 절차적 생성의 냄새만 풍길 뿐이다.

신경을 쓰지 않는다면 계속해서 같은 것만 보게 될 것이다. 예를 들어 <Ugly Baby>에서, 아래 두 레벨은 각각 약 15 초간은 재미있겠지만, 그 지점 이후로는 지루해진다.



각각의 레벨은 5 분 정도 길이였는데, 이것은 우리가 한 레벨을 진행하는 동안 수십 번 변화를 주고 싶어했다는 뜻이다. 보통의 솔루션은 주기적으로 알고리즘을 교체하는 것이지만, 충돌이 일어날 우려가 있었다 -- 경계에서 날카롭게 툭 끝나버리는 PCG 숲을 상상해보라. 다른 방법은 만들어 나가면서 새 알고리즘의 비율을 늘려나가는 방법으로, 점진적으로 사물이 변해가도록 하는 방법이다. 하지만 우리는 다음에서 다룰, 복잡성의 문제에 봉착하게 된다.

3. PCG 알고리즘으로 표현할 수 있는 것이 많아지면서, 아쉽게도 설계하기는 점점 어려워지고 있다.

간단한 레벨을 생성하는 알고리즘을 만드는 것은 정말 쉬운 일이다 -- 하지만 우리가 게임을 복잡하게 만들게 되자 아쉽게도 점점 실행이 어려워졌다. <Ugly Baby>의 예를 들어보자. 우리는 맵 상에 빌딩을 뿌려놓는 스크립트를 성공적으로 만들어낸 경험이 있었다. 그러나 이것을 더 많이 하면서 내부적으로 이런 대화가 오갔다:

"근사해 보이네. 그럼 이걸 단지로 묶어놓으면 어떨까?"

"좋아. 하지만 너무 평범하니까, 좀 섞어 보죠."

"점수판, 터널, 움직이는 플랫폼, 선풍기 날 같은 것도 추가해 봅시다."

"헉, 점수판이 빌딩을 가로막고 있네요. 조금 옮겨보세요. 지루하니까 움직이는 플랫폼 앞에 있을 때 말고는 선풍기 날이 터널 중심에 위치하게 하시고요.

"어, 근데, 아무 것도 안 먹혀요"

마치 초보 게임 개발자가 풀밭을 걸어가는 3D 캐릭터를 보고 "와, 나 MMORPG 만들었다! 쉽네!"라고 말하는 것과 같은 모양새이다. 우리는 간단한 스크립트가 흥미로운 결과를 도출했을 때, 이와 비례해 더 근사한 것을 만들어 내려고 이것을 끝도 없이 확장하려고 한다. 재미있는 구조물은 이를 지탱하는 메커니즘와 신경써서 만든 예외가 필요해지기 시작했다.

4. 재미있는 PCG 는 종종 1)바보같거나 2) 지루한 콘텐츠를 만들어낸다.

가장 간단한 무작위 이름 생성기는 다음과 같이 나타낼 수 있다:

```
# Generate random letters, yo:

for i in 1..random_number():

    name += random_ character()
```

여러 번 반복하면 끝내주는 판타지나 SF, 혹은 아기의 이름을 생성해내는 게 가능할 수도 있다. 아마 "캡틴 락 맥스팩타큘러" 와 같은 멋진 이름을 만들어낼 수도 있을 것이다. 하지만 대부분은 쓰잘데기없이 "ergihwe`=-ufaw38o72wenufse," 같은 이름들만 만들어낸다.

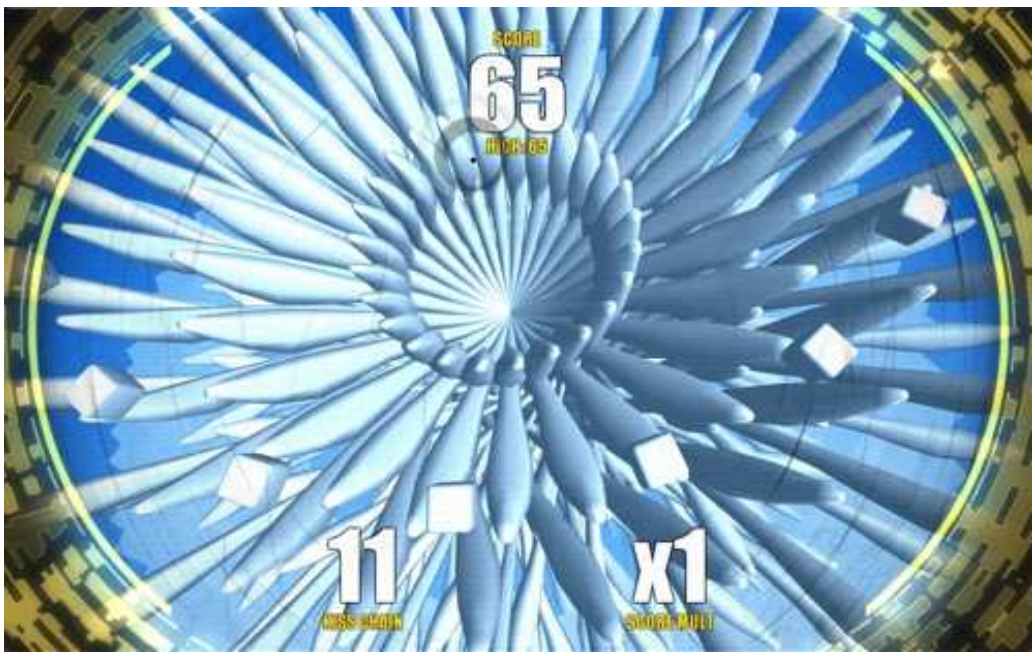
<Ugly Baby>를 만들 때에도 비슷했다. 중심축 주변에 무작위로 조각들을 뿌리는 불확정성을 띤 알고리즘이 비교적 간단하고 쓸만한 터널을 만들어냈다. 이 알고리즘은 다음과 같다.

1. 라이브러리에서 큐브, 삼각형, 곡선과 같은 3D 모델을 무작위로 선택한다.
2. 임의의 숫자 N 과 M 을 선택한다.
3. 고르게 분포한 N 개의 고리를 만들고, 각각의 고리에는 M 개의 모델을 배치한다.
4. 1 번으로 돌아간다.

실행 결과는 이러했다:

시각적으로도 흥미롭고, 또한 실제로 우리에게 유용했다 -- 터널이 만들어진 것이다! 같은 스크립트를 다시 돌렸을 때는 털 같기도 하고, 손가락 같기도 한 다음과 같은 재미나는 터널이 만들어졌다:

이 또한 흥미로울뿐더러 상상 외의 결과였고, 또 유용했다. 플레이어가 날아서 통과할 수 있는 공간이 만들어진 것이다. 세번째 시도를 보자:



이번엔 빽빽한 다각형이 프레임을 막아서서 통과가 불가능한 길이 만들어졌다. 보기 좋고, 다른 어딘가에선 잠재적으로 유용할 수 있겠지만 터널은 아니었다.

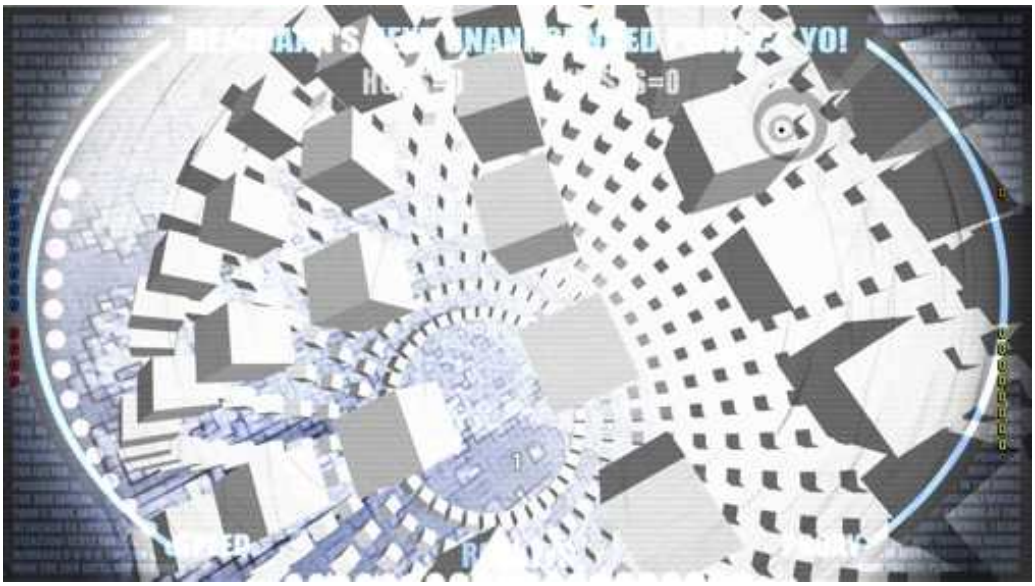
이에 대한 해결책 중 하나는 변수를 통제하는 것이었다. 앞서 말했던 이름 생성기라면, 아마 문법구조(자음+모음+자음+모음+자음)를 만들던가, 흔한 이름과 성의 목록을 만들고 그저 이것들을 조합하는 것이다("Billy Margaret Smith"처럼 말이다). 이처럼 터널을 만들땐 특정한 부피 이하가 되게끔 터널을 만들거나, 사용되는 다각형의 총 개수에 제한을 거는 식이다.

이러한 방식의 (부차적) 문제로 이러한 것들을 꼽을 수 있다:

- 더는 우리를 깜짝 놀라게/기쁘게 하던 재미나는 결과를 볼 수 없다 (안녕, 캡틴 락 맥스팩타큘러).
- 알고리즘의 많은 부분이 특수 사례로 구성되어, 관리가 불가능하게 된다.

레벨 구성을 위한 모듈식 접근

우리는 지난 3 년간 <Ugly Baby>를 만들어왔지만 앞서 말한 문제들을 해결하지 못했다. 하지만 복잡한 결과물을 만들어내기 위해 간단한 것들을 모듈식으로 만들어 조합하는 방식으로 얼마간의 성공을 거둘 수 있었다.

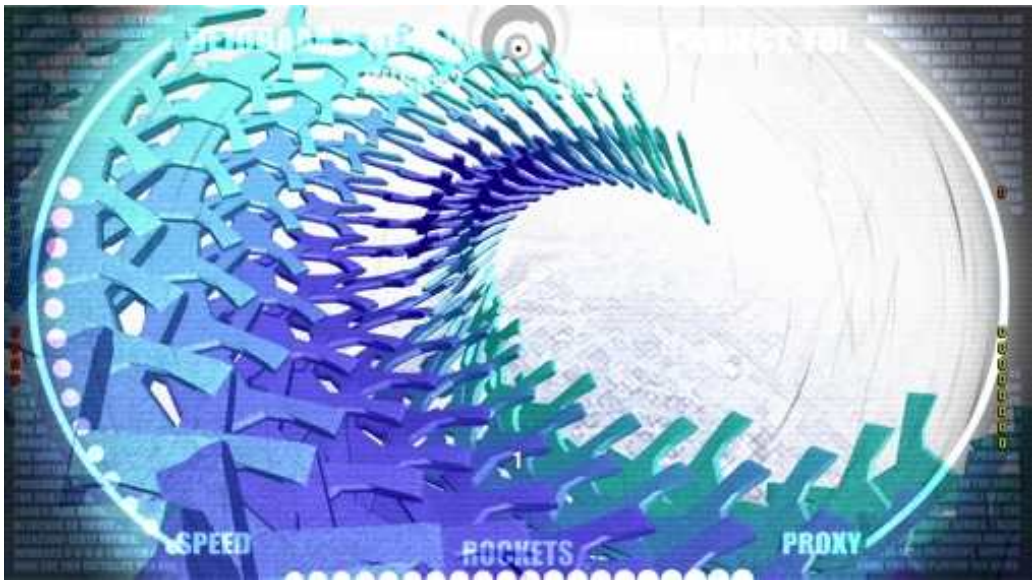


<Aaaaa!>의 레벨 골격을 프로그램 방식으로 만들며 거둔 초기의 성공은 짜릿한 것이었고, 그래서 우리는 기초로 되돌아가기로 했다. 사실 간단한 구조를 만들고 이를 반복해서 시각적으로 흥미로운 것들을 만들어내는 것은 매우 쉽다. 정육면체로 구를 만들어내는 방법을 예로 설명하겠다.

알고리즘을 복잡하게 만드는 대신, 결과물을 반복하여 수정하는 방식을 택하였다:

1. 정육면체로 구를 만들어 낸다.
2. 색을 적용하고 (예를 들어, 중심축을 기준으로 한 물체의 위치에 따라 채도를 정하고 색조를 고른다.)
3. 큐브를 십자가로 바꾼다.
4. 구면상의 특정 위치에만 물체를 배치한다.

최종 결과물은 전혀 구라고 할 수 없는 것이었다:

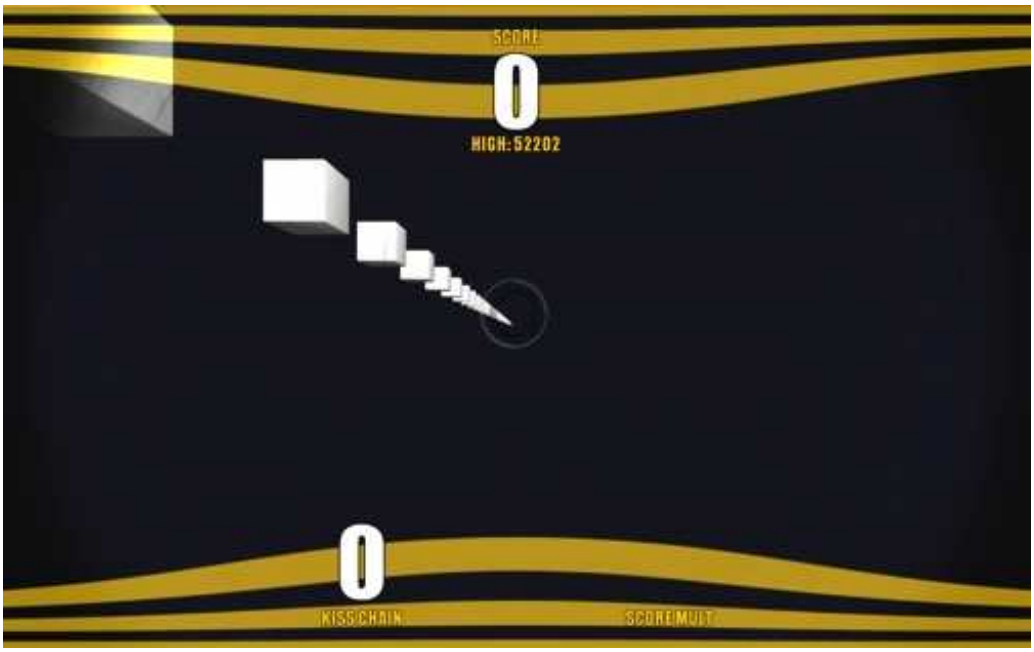


이 간단한 구상에 세 가지 재미난 결론을 얻을 수 있었다. 첫째는 우리가 미묘하게 다른 변수를 투입하는 것으로 너무나 확연히 다른 기하학적 구조를 얻을 수 있다는 점이다. 둘째는 <Ugly Baby>에서 이러한 변수와 오디오 스트림을 결합하여 레벨이 더욱 확연하게 드러날 수 있도록 했다는 점이다. 마지막으로 수정 경로는 기본 구조와 독립적이었고, 그래서 우리는 쉽게 위의 것들을 정육면체 격자 배열에 적용시키고, 또 참신하고 (아마) 유용한 것을 도출해 낼 수 있었다.

이것은 그럴싸한 방법이었기 때문에 우리는 이것을 공식화하기로 했다. <Ugly Baby>의 레벨 생성기는 세 가지의 다음의 모듈로 구성된다:

- **시퀀서(Sequencer)**는 구, 기둥, 그리드, 원통 등등의 대상을 만들어내는 모듈이다.
- **셀렉터(Selector)**는 물체의 한쪽 면이라던가 빵 상자보다 더 큰 물체 등과 같이 어떠한 조건을 충족시키는 대상을 걸러내는 역할을 한다.
- **뮤테이터(Mutator)**는 특성에 따라 대상에 변화를 적용한다. 포지션에 따라 색깔을 바꾸거나 방향에 따라 비율을 바꾸는 등의 예가 이에 해당한다.

여기에 이 모두를 적용시켜보았다:



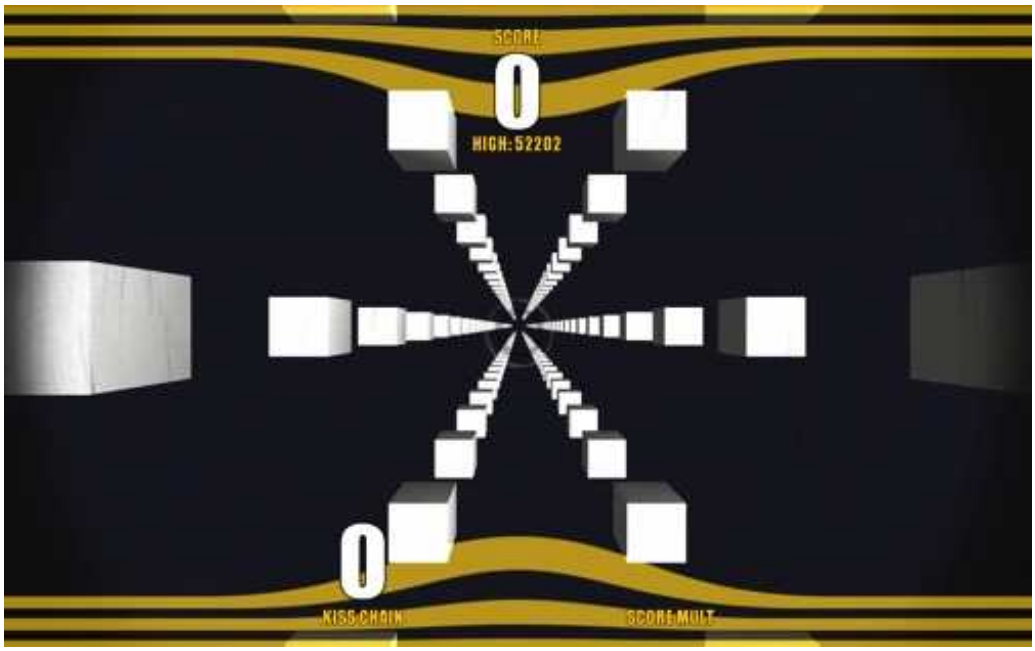
1 단계: 플레이어가 선형의 경로를 따라 비행한다. 낙하하는 축을 따라 블록으로 일직선의 기둥을 만들어 보자.

코드:

```
# Instantiate the column:

sequencer_column = sequencer.Column()

queue = sequencer_column.iterate()
```



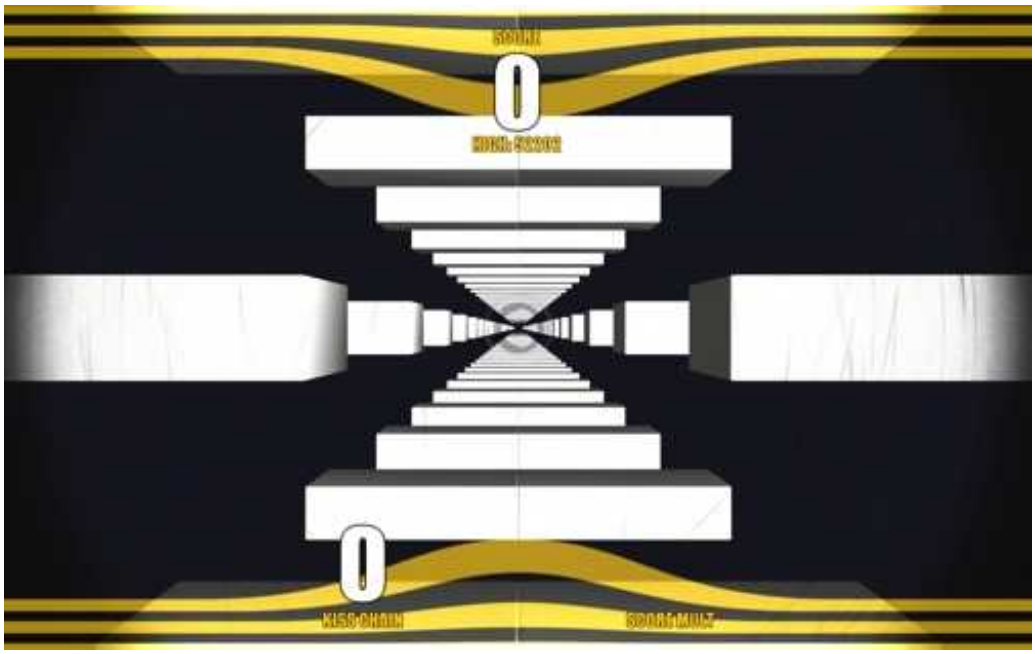
2 단계: 이 레벨은 레일슈터와 비슷하다. 그러므로 뭔가 좀 더 터널처럼 보이는 것을 만들어보자. 하나였던 기둥을 (본질적으로 원통의 모서리인) 여섯 개의 기둥으로 바꿔보자. 여기에 블록 사이의 수직 거리와 축을 둘러싼 블록 기둥의 개수 같은 기본적인 변수를 몇 가지 사용한다.

코드:

```
# Instantiate the cylinder:

sequencer_cylinder = sequencer.Cylinder(layer_delta=4, blocks=6)

queue = sequencer_cylinder.iterate()
```

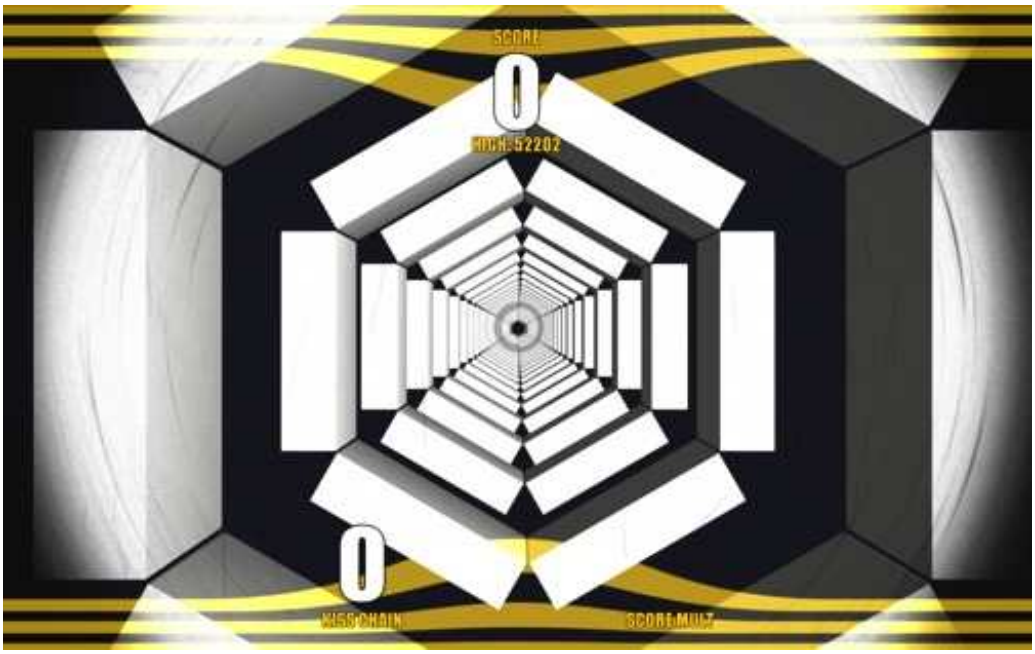


3 단계: 시퀀서가 생성해 낸 모든 조각에 규모를 부여한다.

코드:

```
# Change every piece's scale:

mutator.scale(queue, [1, 4, 1])
```



4 단계: 뮤테이터 노드를 써서 조각들이 Z 축을 향하도록 방향을 재배치하고 다른 모든 조각에 적용하였다.

코드:

```
# Turn pieces to face the player's falling (z) axis:

mutator.face_axis(queue)
```

5 단계: "N 번 마다" 셀렉터 노드는 투입된 조각들 중에서 N 번째의 조각들을 골라낸다. 우리는 뮤테이터를 사용하여 네 조각 마다 하나씩을 빨간 색으로 칠하고 싶었다.

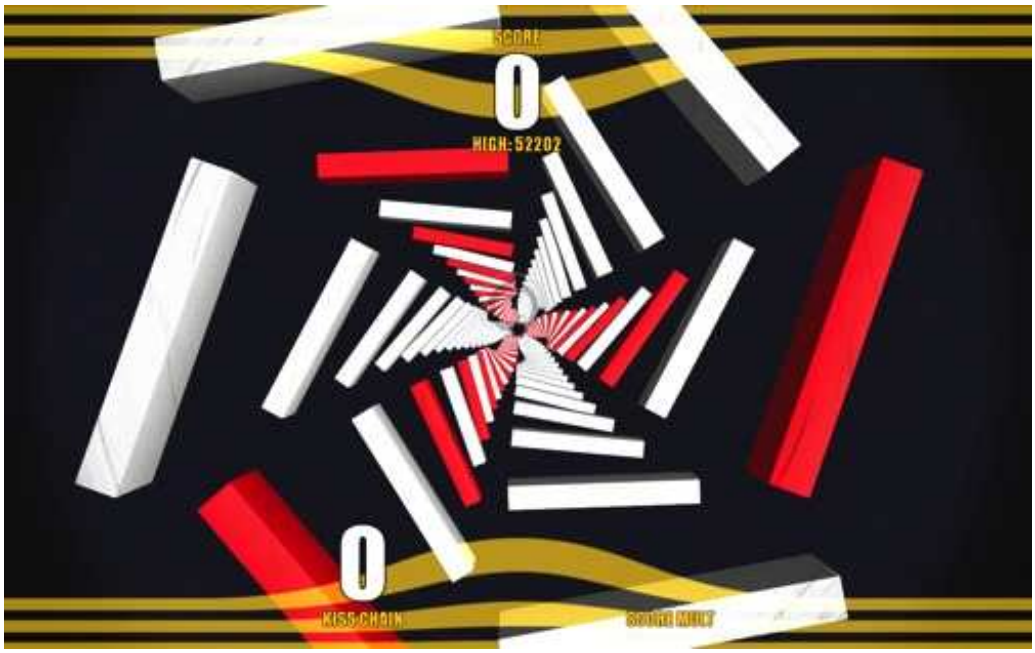
코드:

```
# Get a list of every 4th pieces that comes into the queue:

every_4th_piece = selector.every_n(queue, 4)

# Turn those pieces reddish:

mutator.set_color(every_4th_piece, [255, 32, 0])
```



6 단계: 이제, 수직 방향의 사인 곡선 모양으로 이것들을 배치하자.

코드:

```
# Pan from -45..45 depending on a piece's position along the player's falling axis:

mutator.cyclic_rotate(queue, freq=0.1, low=[-45, 0, 0], high=[45, 0, 0])
```

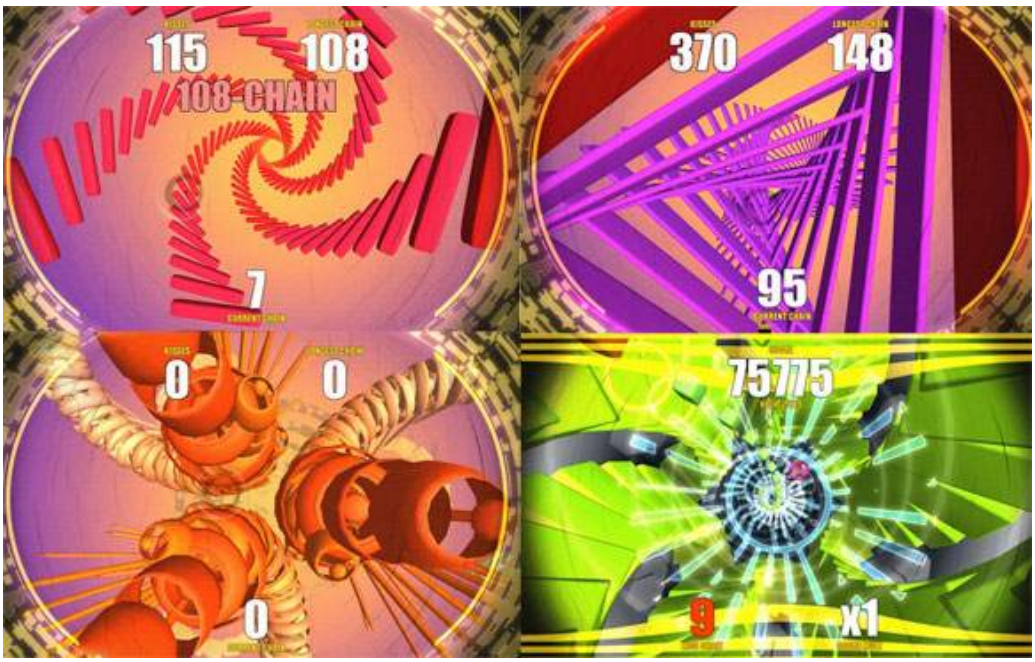
여기서, 우리는 수많은 것들을 할 수 있었다.

1. 각각의 효과가 서로 분리돼 있었으므로, 우리는 여러 가지를 넣고 빼며 바꿔볼 수 있었다. 이런 조립성은 우리가 기존의 것을 대신할 만한 새로운 것을 시도해 볼 수 있도록 하였다.

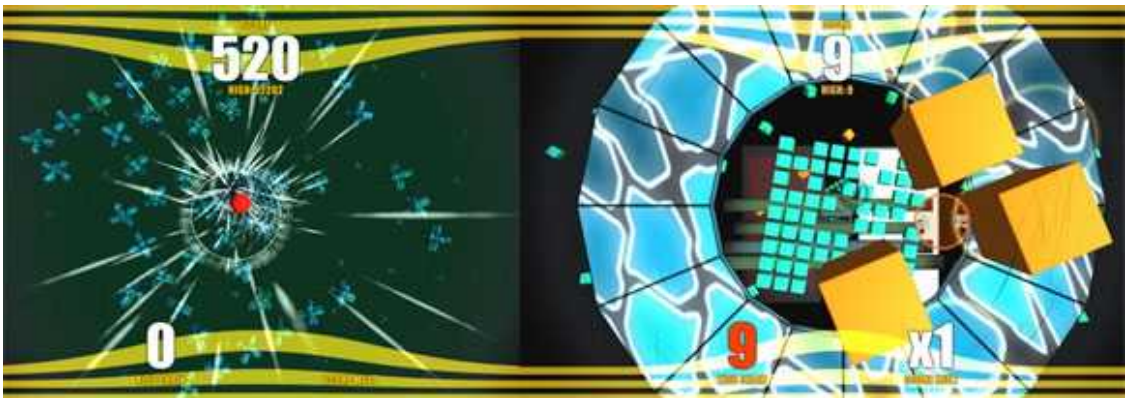
2. <Ugly Baby>는 시각과 음악의 결합에 의한 게임이다. 우리는 음악에 따라 플레이어가 어떻게 느끼는지를 알고 있다는 장점을 활용하여 청각 신호를 면밀히 고려하였고, 이에 기반한 구조를 만들 수 있었다.

3. 우리는 플레이어들이 약간의 조작을 가하여 자신만의 레벨을 만들 수 있도록 허용하였다. 기둥이 여섯 개가 아니라 두 개라면? 혹은 모든 조각이 아주 커다랗다면?

뮤테이터 변수 안에서의 작은 변화는 레벨 안에서 받아들일 수는 있지만 분명히 존재하는 변화를 만들어냈다. 여기에, 기둥으로 구성된 기본 모델과 마찬가지로 우리가 숫자와 스케일을 조작하여 만들어낸 변형을 보자.



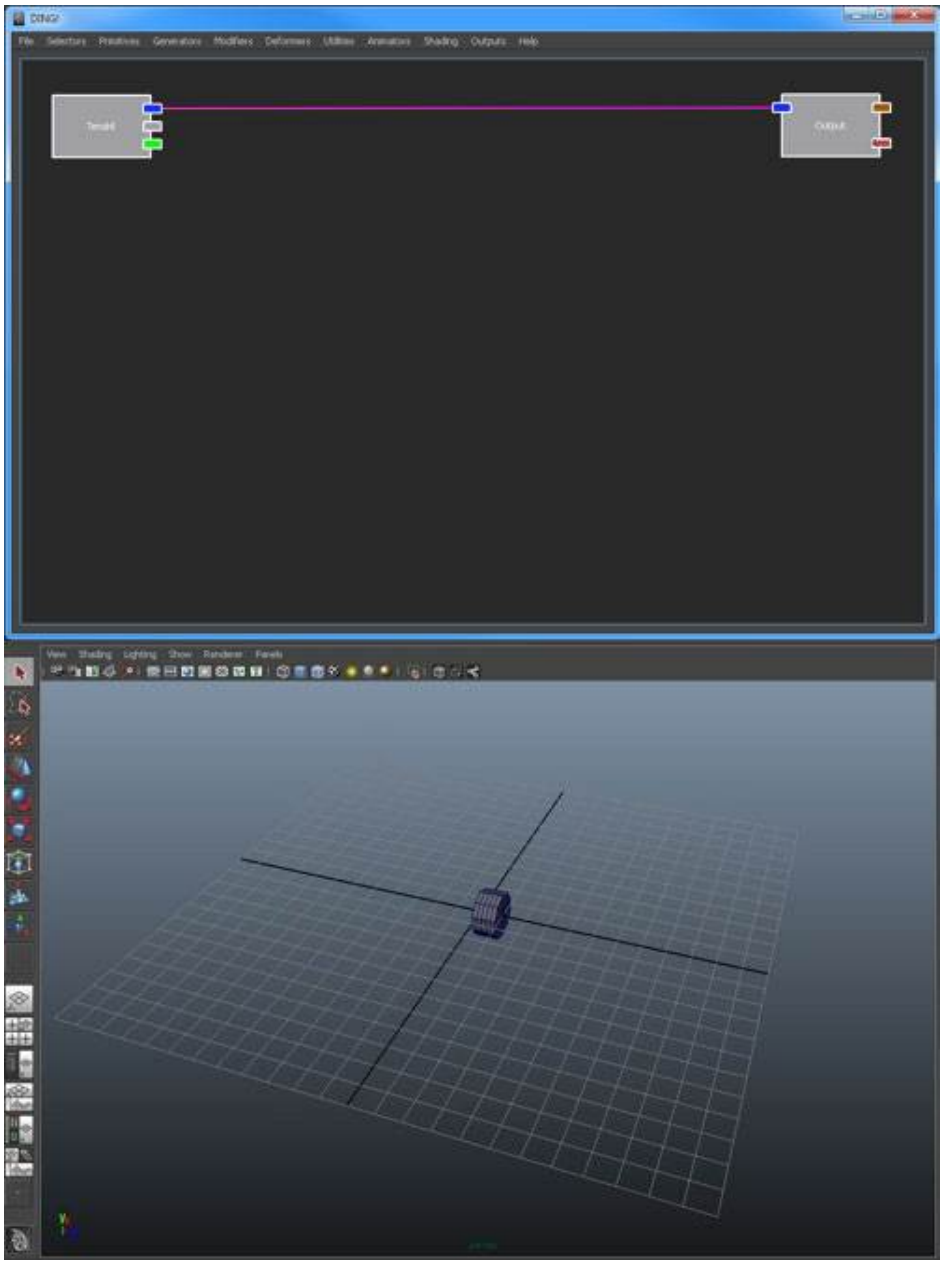
이후로 기본 어휘(대상을 격자로 만듦, 아래 왼쪽)로 취급하여, 조합(격자에 고리 모양을 더함, 아래 오른쪽)을 했다



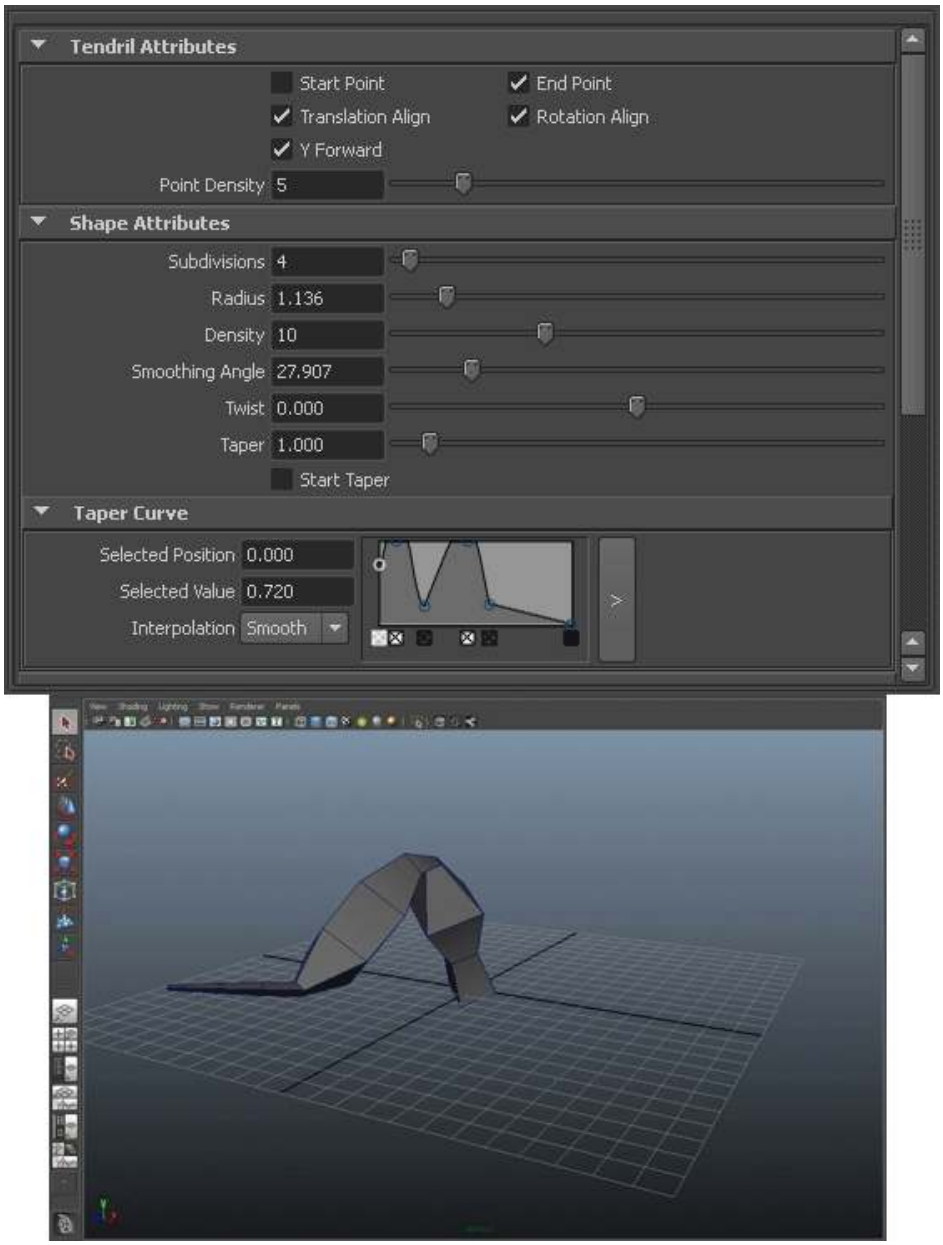
이러한 것들은 추상적인 구조에서 아주 잘 돌아갔지만, 우리는 이것이 더 유기적인 구조에서도 먹히는지를 알고 싶었다.

유기적 모델 창조에서의 모듈

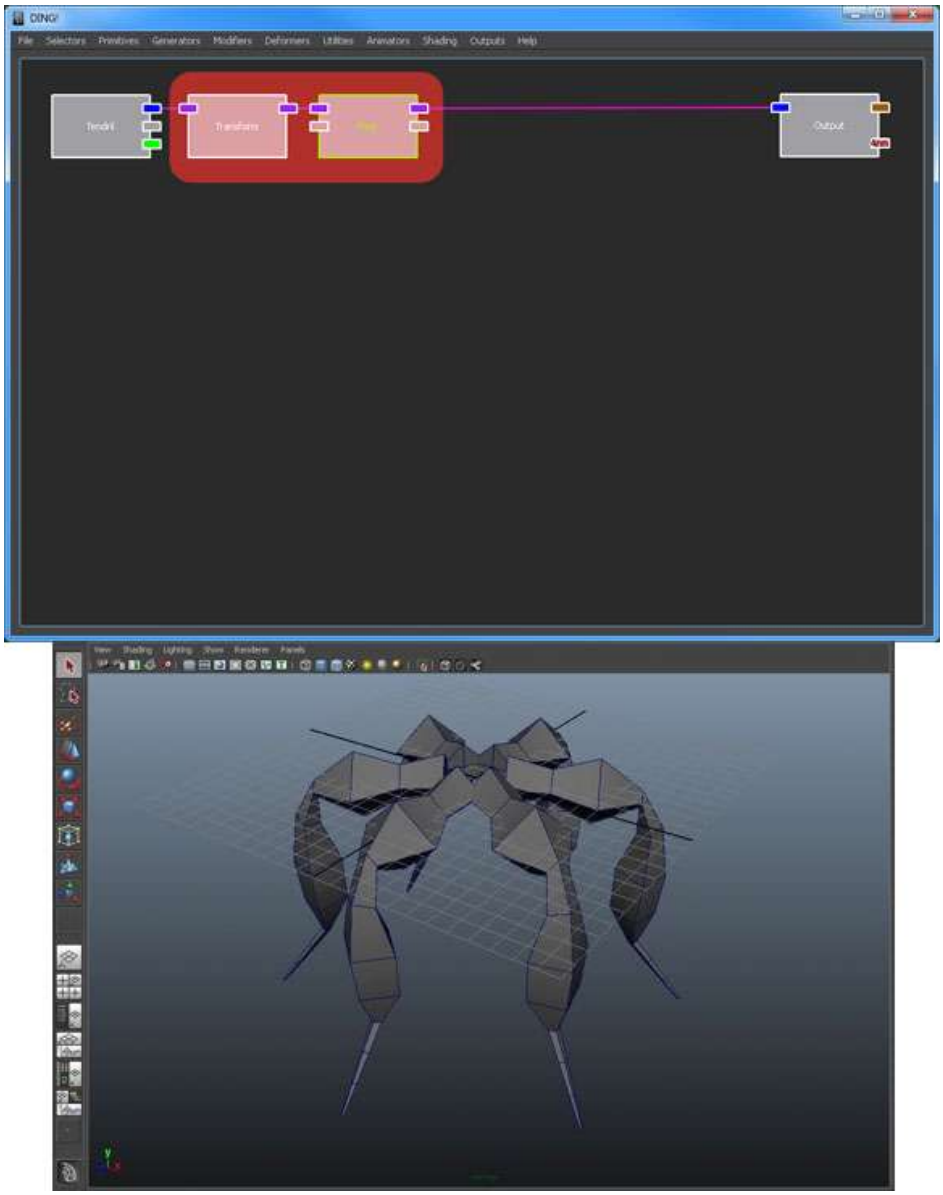
우리는 더 유기체적인 모델을 가지고 <Ugly Baby>의 추상적이고 기하학적인 레벨을 만들어냈다. 다시 말해서 우리는 Maya 로 DING 이라는 그래픽, 노드 기반의 유틸리티를 만들어낸 것이다. 이런 환경 안에서 곤충을 만들어내는 과정을 살펴보자.



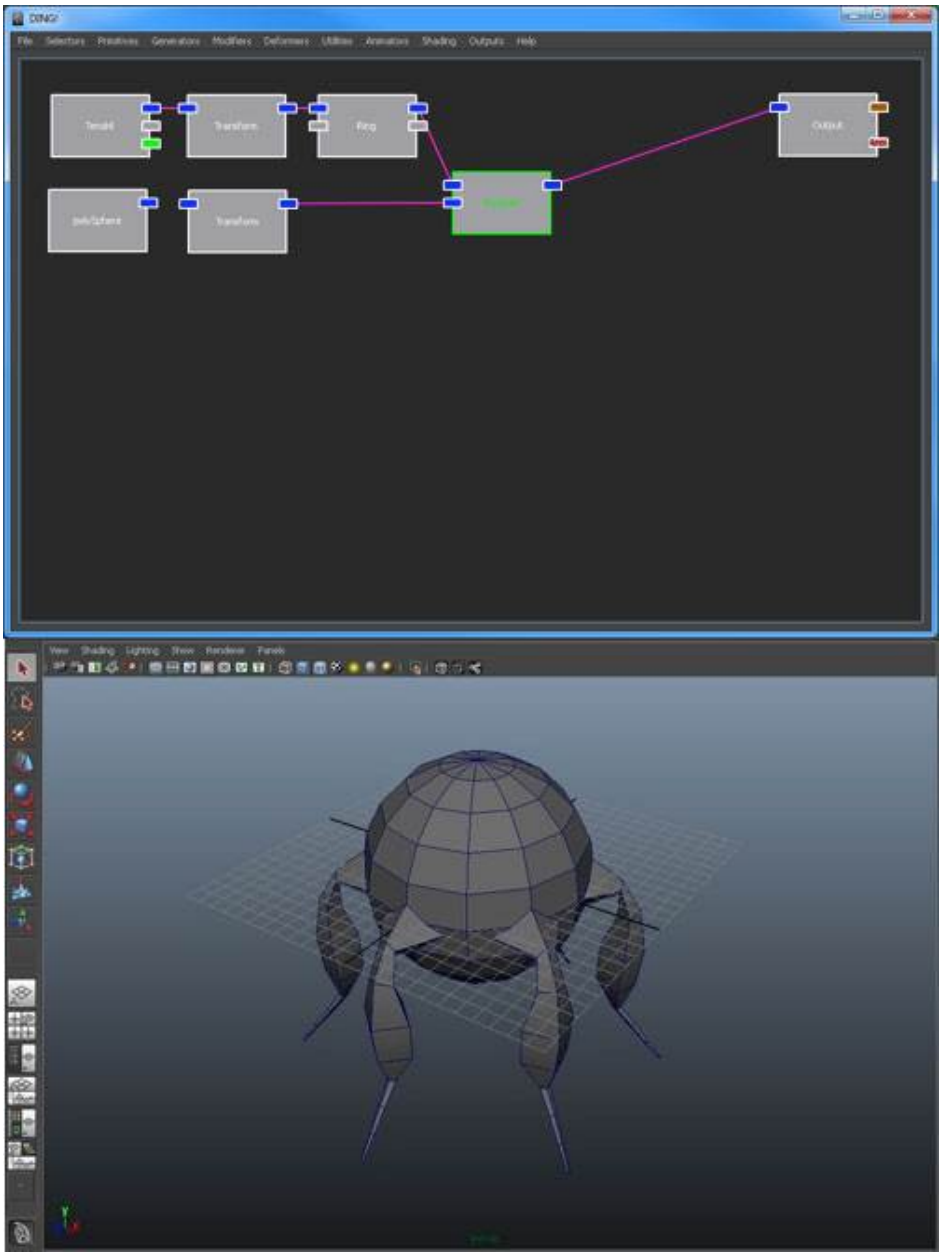
원통을 만들어내는 것으로 시작한다. 위 두개 의 스크린 샷 -- 위는 그래프를 나타내며(왼쪽 노드는 실린더를 만들어내며, 오른쪽 노드는 이를 나타내준다), 아래 이미지는 그 결과를 나타낸다. DING 은 기하학적인 모양을 만들어내며, Maya 는 이를 나타내 준다(그리고 나중에 FBX 로 내보기도 한다).



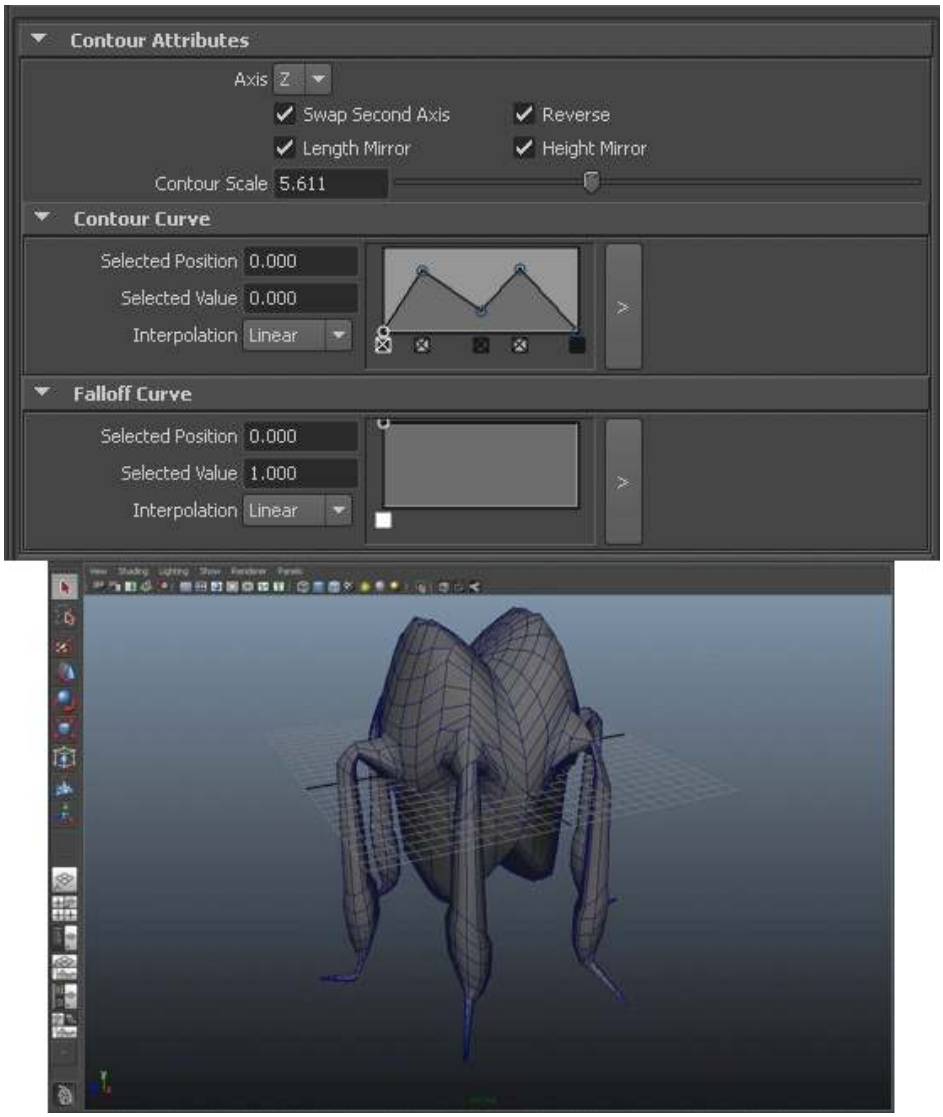
그 다음에, 어디선 더 통통하게 만들고, 어디선 더 가늘게 만들어가며 원통이 점점 가늘어지게 (너무 과격하진 않게) 만든다. 눈으로 보이는 그래프는 점점 가늘어져가는 물체의 윤곽이다 -- 이것을 통통하게도 만들었다가, 관절을 잡아당겼다가, 끝에 가선 가늘어지는 발이 되도록 수작업으로 만드는 것이다.



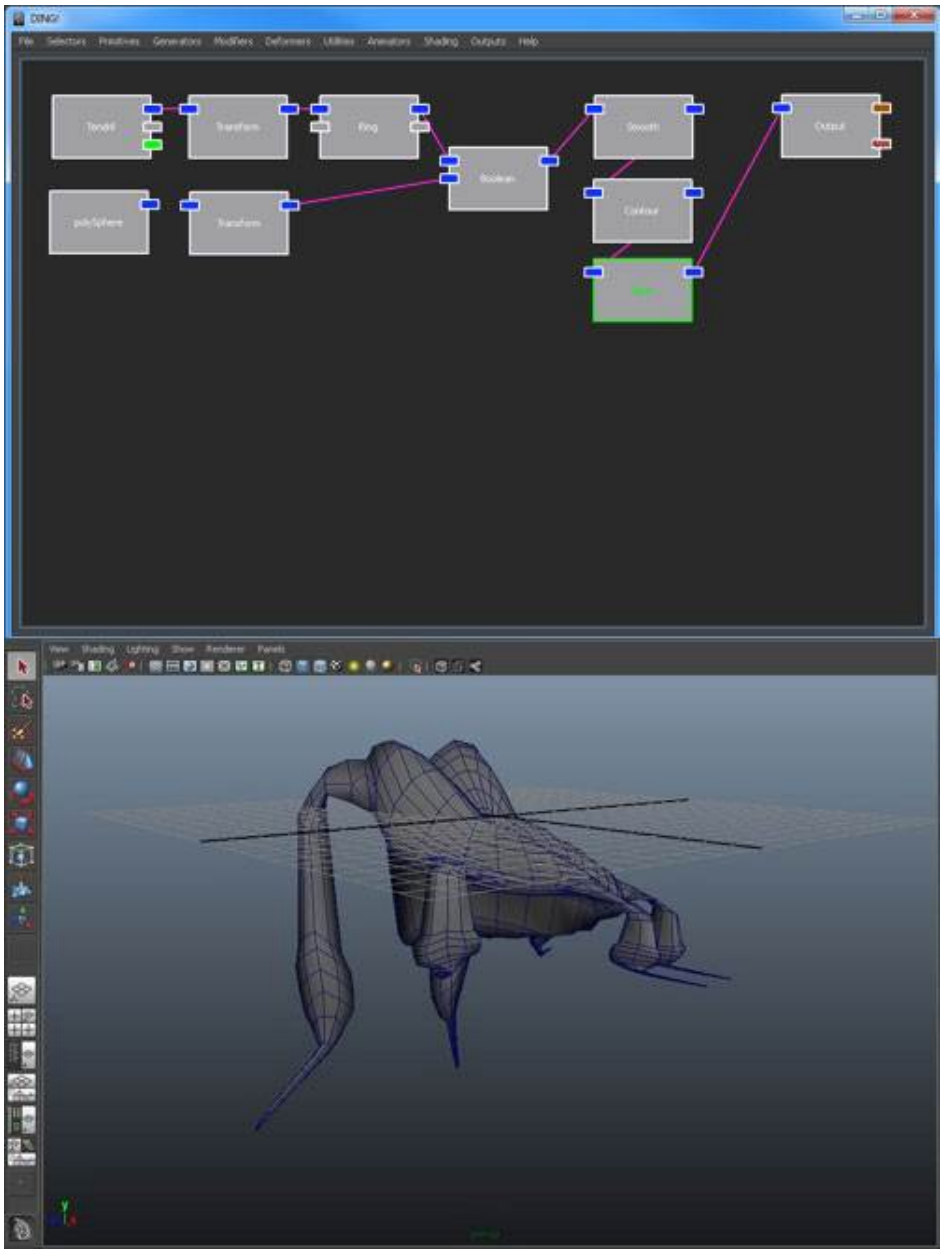
우리는 수직방향으로 조각들을 돌리기 위한 노드(빨간색 부분)을 버리고, "고리"노드를 더해 형태를 6 개의 고리 모양으로 만들었다.



구 노드를 더하고, 이를 고리와 합쳤다(합집합 노드). 이제 거미와 비슷하게 보이기 시작한다.



앞서 다리를 가늘게 뺐으므로, 이제 전체에 윤곽 노드를 더하였다. 이것이 가운데가 움푹 들어가도록 몸통을 죄었다. 우리는 더 매끄럽게 만들기 위해 격자망을 더 잘게 나누었다.



이제 수직의 축을 따라 윤곽을 잡아 가며 앞을 짜부라뜨리자. 이 모습은 뭔가 진드기 비슷한 것처럼 보인다.

모델을 런타임에 만든 것이 아니므로, 이것은 레벨 구조를 이루는 음악에 반응하지도 않는다. 그러나 이것은 우리의 두 가지 목표를 충족킨다. 하나는 흥미로운 콘텐츠를 빨리 만들 수 있게 해준다는 점이고 (이 방법은 Maya 에 익숙하지 않은 우리 팀원도 쉽게

습득했다), 다른 하나는 커다란 실험을 가능하게 해 준다는 점이다. 이러한 조작이 파괴적이지 않으므로 우리는 다리와 고리의 갯수를 늘리거나 줄여볼 수도 있고, 윤곽을 바꿔볼 수도 있으며(아래), 심지어 원통과 구 등의 요소를 바꿔가며 어떤가를 살펴볼 수도 있었다. 우리가 이번 크리스마스에 받고 싶은 선물은 "무작위(randomize)" 버튼이다.

어떤 목적을 위한 것이든 그래프 기반의 툴이 존재한다는 사실을 명심하기 바란다. <Ugly Baby>의 추상적인 레벨 디자인과, 적과 같은 유기체를 만들어냈지만, 우리는 2011 년 작 <Aaaaa!> (Awesome 의 AaaaaAAaaaAAAaaAAAAaAAAAA!!!라고 일컬음)에서 진짜 같은 질감을 원했고, 그래서 Spiral Graphics 의 훌륭한 노드 기반 툴, Genetica 를 사용했다.

결론

"모듈화"는 새로운 것이 아니며, 노드를 조합하는 것이 레벨 디자인을 할 때 대상이 겹친다던가, 말도 안되는 결과물이 나온다던가 하는 문제를 해결해준 것도 아니다. 하지만 이것은 더 생산적이었으며 유용한 결과물을 내놓으면서 동시에 쓰잘데없는 곁가지를 쳐주기도 한다.

그러므로 PCG 가 가진 이점(더 빠른 콘텐츠 생산, 역동적 콘텐츠, 더 작은 용량, 반가운 우연에 의한 창의성 증대)을 유지하며 이것의 취약점을 줄여가는 방식(코드가

얽히고설키지 않게 하면 좀 더 관리가 쉬워진다든지)으로 활용 가능한 것이다. PCG 가 만병통치약은 아니지만 우리는 이를 이용하여 다른 방법으로는 불가능한 콘텐츠를 성공적으로 만들어낼 수 있다.

바로 그거다! 우리가 비록 PCG 전문가는 아니지만, 여러분의 하늘을 나는 자동차를 향한 여정에 이 글이 유용한 자료가 되었기를 바란다.