Chapter 3:
Geometry

Topics:

Part I: Geometry design, space locator creation, vertex creation, vertex proportion scaling

Part II: surface creation, polygon surface uniting, reverse polygon normal, geometry creation

Chapter 3:
Geometry

เอกสารประกอบการเรียน
รายวิชา ANI 951301
สาขาวิชาแอนนิเมชันและเกม
วิทยาลัยศิลปะ สื่อ และ
เทคโนโลยี
มหาวิทยาลัยเชียงใหม่

951301: Advanced Animation and Rendering Techniques

วัตถุประสงค์

1. เพื่อให้นักศึกษาเข้าใจพื้นฐานการขึ้นรูปด้วย MEL scripts
2. อธิบายพื้นฐานการสร้างจุด vertex ขึ้นมาด้วยการกำหนด position ให้กับจุดเหล่านั้น
3. สามารถสร้างระนาบ (faces) ขึ้นมาจากการเชื่อมต่อจุดใดๆ ตั้งแต่สามจุดขึ้นไป
4. เข้าใจการเชื่อมต่อระนาบใดๆเพื่อใช้ในการสร้างรูปทรงใหม่ที่ไม่สามารถหรือทำได้ยากด้วยวิธีแบบปกติ

เนื้อหาการสอน

ส่วนที่ 1: การออกแบบรูปทรง geometry, การสร้างและการจัดวางตำแหน่ง space locator, การสร้างจุด, และการปรับสัดส่วนขนาดของจุดแบบเป็นกลุ่มก้อน

ส่วนที่ 2: การสร้างพื้นผิวให้กับวัตถุจากการเชื่อมจุด, การรวมพื้นผิวเพื่อสร้างรูปทรง, การกำหนดทิศทางการประมวลผลให้กับวัตถุ, และการสร้างรูปทรง geometry

Geometry

ความสามารถที่สำคัญอีกอย่างของ MEL scripts คือการขึ้นรูปและการแก้ไขวัตถุ (geometry) ในการขึ้นรูปนั้น เรามีความจำเป็นที่เราต้องเข้าใจพื้นฐานทางคณิตศาสตร์ที่เกี่ยวกับการขึ้นรูปทรงอยู่บ้างเช่นเรื่องของ algebra, ค่า sine, cosine, tangent ของสามเหลี่ยม, การคำนวณคณิตศาสตร์แบบ vector เป็นต้น

การขึ้นรูปทรงพื้นฐานถือเป็นจุดเริ่มต้นในการประยุกต์ใช้โปรแกรม Maya ในการใช้ MEL ก็เช่นเดียวกัน เราจะสังเกตได้ว่าโปรแกรม Maya มีรูปทรงพื้นฐาน (polygon primitives) ให้เริ่มต้นทำงานอยู่หลายประเภท เช่น cube, sphere, cylinder, torus หรือ cone แต่ยังขาดอีกรูปทรงหนึ่งนั่นคือ Polyhedron ซึ่งมีรูปทางคล้ายกับ sphere แต่จะมี faces ที่มีขนาดเท่ากันทุกด้าน วันนี้เราจะมาสร้าง tool ในการสร้างรูปทรงนี้กัน

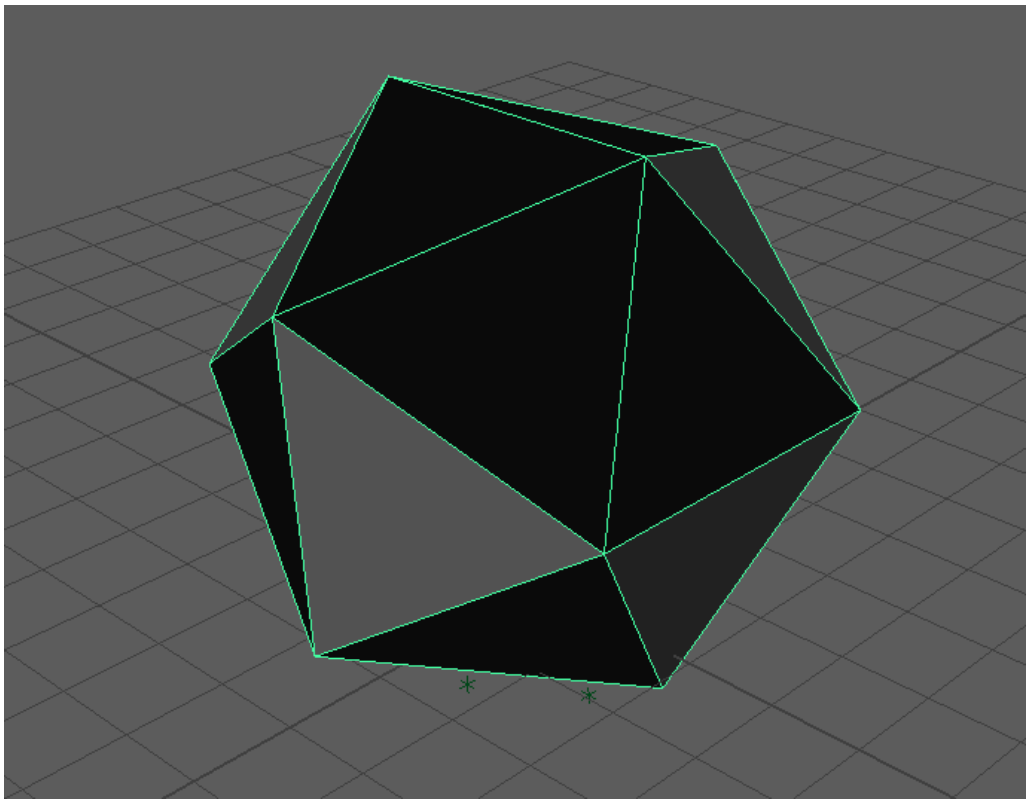

Fig 03-01: ตัวอย่างรูปทรงแบบ polyhedron ที่ประกอบขึ้นจากพื้นผิวสามเหลี่ยมด้านเท่า

ภาพด้านหน้าคือตัวอย่างของ polyhedron จะสังเกตได้ว่ามันถูกสร้างขึ้นมาจาก faces สามเหลี่ยมด้านเท่า ซึ่งจะมีขนาดเท่ากันหมดทุกด้านทั้งด้านและมุมของสามเหลี่ยม ซึ่งเราไม่สามารถสร้างโดยการดัดแปลง polygon primitive sphere ที่โปรแกรมมีให้ได้ เนื่องจากต้องใช้หลักการทางคณิตศาสตร์ที่แตกต่างกันนั่นเอง

951301: Advanced Animation and Rendering Techniques

บทเรียนนี้จะพยายามใช้วิธีที่เข้าใจได้ง่ายที่สุดในการทำ จุดมุ่งหมายเพื่อให้นักศึกษาเข้าใจถึงการขึ้นรูปขึ้นมาใหม่ทั้งหมด โดยเริ่มจากการสร้างจุด แล้วสร้างพื้นผิวจากการเชื่อมต่อชุดของจุด แล้วจึงสร้างรูปทรงจากการเชื่อมต่อชุดของพื้นผิว ทีนี้เรามาทดลองสร้างรูปทรงนี้กัน

Workshop 1

ขั้นตอนแรกเราลองมาทำความเข้าใจถึง design ของ polyhedron กันก่อน polyhedron ที่เราจะทดลองทำกันในวันนี้ประกอบด้วยพื้นผิวทั้งหมดจำนวน 20 faces โดยทั้ง 20 faces จะถูกสร้างขึ้นมาจากการเชื่อมต่อจุด (vertex) ทั้งหมด 12 จุด โดยทั้งหมดจะถูกประกอบกันเข้าเพื่อสร้างรูปทรงคล้ายกับทรงกลม ซึ่งหลักการทางคณิตศาสตร์ที่จำเป็นต้องใช้มีเพียงหลักการเดียวคือหลักของ golden ratio ในการสร้างสามเหลี่ยมด้านเท่า ซึ่งเป็นหลักการที่ Leonardo da Vinci ใช้สร้าง polyhedral ขึ้นมาเมื่อปี ค.ศ. 1509 นั่นเอง ซึ่งการทำงานของเราใน workshop นี้จะมีขั้นตอนดังนี้

1. การหาตำแหน่งให้กับจุด vertex ทั้งสิบสองจุด
2. การสร้างระนาบ (faces) โดยการเชื่อมต่อ vertex เหล่านั้น
3. การตั้งชื่อให้กับระนาบที่สร้าง
4. การรวมระนาบทั้งหมดเพื่อสร้างเป็นรูปทรงรูปทรงเดียว

ขั้นตอนแรกคือการหาตำแหน่งให้กับจุด vertex เพื่อใช้ในการขึ้นรูป เราต้องคำนวณหาสูตรที่เหมาะสมในการหา vertex co-ordinates ซึ่งในกรณีนี้เราสามารถใช้หลักการของ golden ratio มาใช้ซึ่งหลักการ golden ratio มีดังนี้

$$(\sqrt{5}-1)/2$$

เนื่องจากพื้นฐานของรูปทรง polyhedron มีพื้นฐานมาจากความสัมพันธ์ของรูปสามเหลี่ยมด้านเท่า จึงสามารถประยุกต์ใช้หลักการ golden ratio มากำหนดหาตำแหน่ง vertex ได้ จากสูตรนี้เราจะสามารถกำหนดตำแหน่งของจุดทั้งสิบสองจุดได้จากสูตรด้านล่าง

(+/- 1), 0, (+/- golden ratio)

0, (+/- golden ratio), (+/- 1)

(+/- golden ratio), (+/- 1), 0

951301: Advanced Animation and Rendering Techniques

ก่อนที่เราจะเริ่มทำงาน เราควรจะ define variable ให้กับค่า golden ratio กันก่อน เพื่อความสะดวกในการเรียกค่าต่อไป ในที่นี้เราจะประกาศตัวแปรแบบ float (เพราะค่า golden ratio เป็นเลขทศนิยม) โดยตั้งชื่อว่า gM ให้มีค่าเท่ากับค่าของ $(\sqrt{5}-1)/2$ ดังตัวอย่างด้านล่าง

```
float $gM = ((sqrt (5)-1)/2);
```

ใน MEL เราสามารถแทนค่าของ square root ได้ด้วยคำสั่ง sqrt ตามด้วยตัวเลขที่เราต้องการ

เมื่อเราได้ค่าตัวแปร $gM เข้ามาในระบบของเราแล้ว ขั้นต่อไปเราจะประกาศค่าตัวแปรให้กับตำแหน่งของ locators ทั้ง 12 อัน โดยค่าที่ได้จะต้องเก็บค่าของตำแหน่งในแกน X, Y และ Z ดังนั้นเราจะประกาศค่าตัวแปรให้เป็นแบบ vector ดังตัวอย่างด้านล่าง

```
vector $v01 = << 1, 0, $gM >>;
vector $v02 = << -1, 0, $gM >>;
vector $v03 = << 1, 0, (-$gM) >>;
vector $v04 = << -1, 0, (-$gM) >>;
vector $v05 = << 0, $gM, 1 >>;
vector $v06 = << 0, $gM, -1 >>;
vector $v07 = << 0, (-$gM), 1 >>;
vector $v08 = << 0, (-$gM), -1 >>;
vector $v09 = << $gM, 1, 0 >>;
vector $v10 = << $gM, -1, 0 >>;
vector $v11 = << (-$gM), 1, 0 >>;
vector $v12 = << (-$gM), -1, 0 >>;
```

โดยที่ $v01 เป็นตัวแปรแบบ vector ที่มีค่าเป็นตำแหน่ง X = 1, Y = 0 และ Z = $gM (ค่าของ golden ratio)

ขั้นตอนต่อไปคือการสร้าง vertex ก่อนที่เราจะเริ่มสร้าง vertex นั้น เราควรจะสร้าง locator ขึ้นมาเพื่อ mark ตำแหน่งทั้ง 12 จุดของ vertex ที่ต้องการสร้างก่อน โดยคำสั่งในการสร้าง locator สามารถทำได้โดย

```
spaceLocator –position 0 0 0;
```

โดยที่ flag –position สามารถย่อได้ด้วย –p และ 0 0 0 คือตำแหน่งในแกน X, Y และ Z ของ locator นั่นเอง ซึ่งในที่นี้เราจะแทนค่าด้วยสูตร golden ratio ที่ให้ไว้ด้านต้น ทีนี้เราลองมาสร้าง locators ทั้ง 12 จุดกันดังตัวอย่างด้านล่าง

```
spaceLocator
-position ($v01.x) ($v01.y) ($v01.z)
-name "Vertex01";
spaceLocator
-position ($v02.x) ($v02.y) ($v02.z)
-name "Vertex02";
```


951301: Advanced Animation and Rendering Techniques

```
spaceLocator
-position ($v03.x) ($v03.y) ($v03.z)
-name "Vertex03";
spaceLocator
-position ($v04.x) ($v04.y) ($v04.z)
-name "Vertex04";
spaceLocator
-position ($v05.x) ($v05.y) ($v05.z)
-name "Vertex05";
spaceLocator
-position ($v06.x) ($v06.y) ($v06.z)
-name "Vertex06";
spaceLocator
-position ($v07.x) ($v07.y) ($v07.z)
-name "Vertex07";
spaceLocator
-position ($v08.x) ($v08.y) ($v08.z)
-name "Vertex08";
spaceLocator
-position ($v09.x) ($v09.y) ($v09.z)
-name "Vertex09";
spaceLocator
-position ($v10.x) ($v10.y) ($v10.z)
-name "Vertex10";
spaceLocator
-position ($v11.x) ($v11.y) ($v11.z)
-name "Vertex11";
spaceLocator
-position ($v12.x) ($v12.y) ($v12.z)
-name "Vertex12";
```


951301: Advanced Animation and Rendering Techniques

เมื่อ execute คำสั่งจะได้ผลลัพธ์ดังนี้

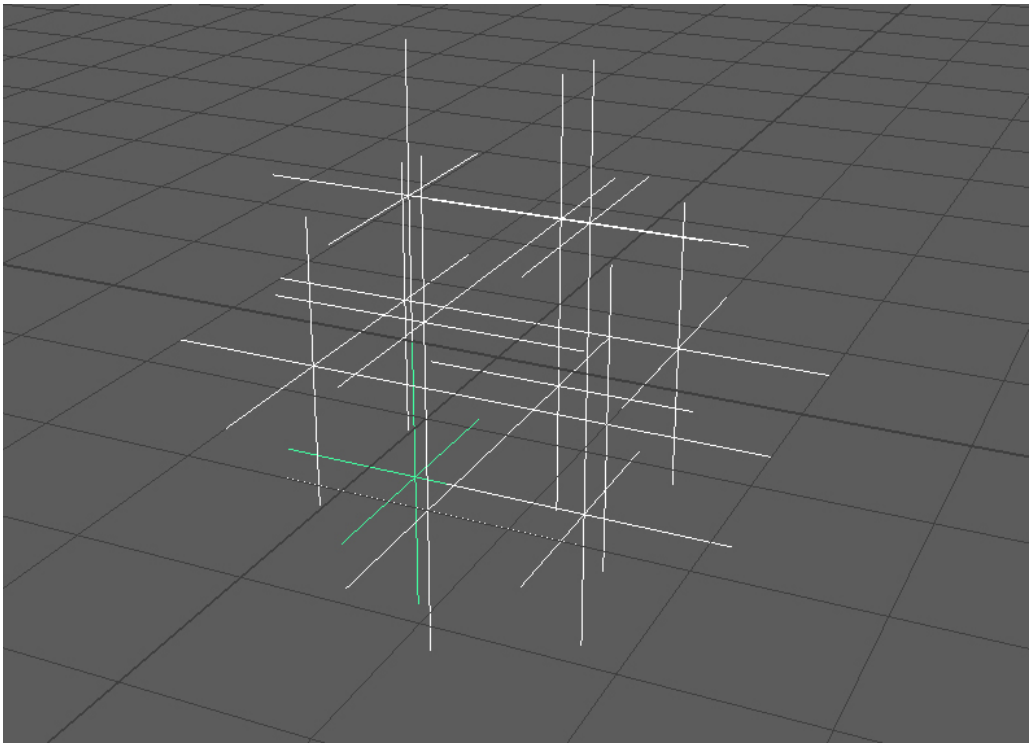

Fig 03-02: แสดงตำแหน่งของ space locator ที่วางซ้อนทับกันอยู่

จะสังเกตได้ว่า locators ทั้ง 12 ตัวที่สร้างขึ้นมามีขนาดใหญ่เกินไปทำให้เราไม่สามารถมองตำแหน่งได้ชัดเจน ในกรณีนี้เราสามารถปรับขนาด locators ได้จากคำสั่ง scale แต่ต้องระบุพิกัดศูนย์กลางการ scaling ไปที่ตำแหน่งศูนย์กลางของตัว locator นั้นๆ เพื่อป้องกันการเคลื่อนที่ออกจากจุดเดิมอันเนื่องมาจากการใช้ตำแหน่ง local มาใช้ในการ scale ซึ่ง flag ที่ใช้คือ –p

```
scale -p ($v01.x) ($v01.y) ($v01.z) 0.1 0.1 0.1;
```

ด้านบนคือตัวอย่างของการสั่งปรับขนาดของ locator ตัว Vertex01 ให้มีขนาดเหลือเพียงสิบเปอร์เซ็นต์ทั้งในแนวแกน X, Y และ Z จากขนาดเดิม โดยค่าในวงเล็บที่ตามหลัง –p จะต้องเป็นค่าของตำแหน่ง locator นั้นๆ คำสั่งที่เติมเข้าไปรวมกับของเดิมที่เรามีอยู่จะเป็นดังตัวอย่างด้านล่าง โดยที่ตัวหนาคือคำสั่งที่เราเพิ่มเข้าไป

```
spaceLocator
-position ($v01.x) ($v01.y) ($v01.z)
-name "Vertex01";
scale -p ($v01.x) ($v01.y) ($v01.z) 0.1 0.1 0.1; // เพิ่มบรรทัดนี้
spaceLocator
-position ($v02.x) ($v02.y) ($v02.z)
-name "Vertex02";
scale -p ($v02.x) ($v02.y) ($v02.z) 0.1 0.1 0.1; // เพิ่มบรรทัดนี้
```


951301: Advanced Animation and Rendering Techniques

```
spaceLocator
-position ($v03.x) ($v03.y) ($v03.z)
-name "Vertex03";
scale -p ($v03.x) ($v03.y) ($v03.z) 0.1 0.1 0.1; // เพิ่มบรรทัดนี้
spaceLocator
-position ($v04.x) ($v04.y) ($v04.z)
-name "Vertex04";
scale -p ($v04.x) ($v04.y) ($v04.z) 0.1 0.1 0.1; // เพิ่มบรรทัดนี้
spaceLocator
-position ($v05.x) ($v05.y) ($v05.z)
-name "Vertex05";
scale -p ($v05.x) ($v05.y) ($v05.z) 0.1 0.1 0.1; // เพิ่มบรรทัดนี้
spaceLocator
-position ($v06.x) ($v06.y) ($v06.z)
-name "Vertex06";
scale -p ($v06.x) ($v06.y) ($v06.z) 0.1 0.1 0.1; // เพิ่มบรรทัดนี้
spaceLocator
-position ($v07.x) ($v07.y) ($v07.z)
-name "Vertex07";
scale -p ($v07.x) ($v07.y) ($v07.z) 0.1 0.1 0.1; // เพิ่มบรรทัดนี้
spaceLocator
-position ($v08.x) ($v08.y) ($v08.z)
-name "Vertex08";
scale -p ($v08.x) ($v08.y) ($v08.z) 0.1 0.1 0.1; // เพิ่มบรรทัดนี้
spaceLocator
-position ($v09.x) ($v09.y) ($v09.z)
-name "Vertex09";
scale -p ($v09.x) ($v09.y) ($v09.z) 0.1 0.1 0.1; // เพิ่มบรรทัดนี้
spaceLocator
-position ($v10.x) ($v10.y) ($v10.z)
-name "Vertex10";
scale -p ($v10.x) ($v10.y) ($v10.z) 0.1 0.1 0.1; // เพิ่มบรรทัดนี้
spaceLocator
-position ($v11.x) ($v11.y) ($v11.z)
-name "Vertex11";
scale -p ($v11.x) ($v11.y) ($v11.z) 0.1 0.1 0.1;
spaceLocator
-position ($v12.x) ($v12.y) ($v12.z)
-name "Vertex12";
scale -p ($v12.x) ($v12.y) ($v12.z) 0.1 0.1 0.1;
```


951301: Advanced Animation and Rendering Techniques

เราจะได้ locators ที่มีขนาดที่เหมาะสมกับการทำงานใน workshop นี้โดยที่ locators ทั้ง 12 ตัวจะมีตำแหน่งดังภาพตัวอย่างด้านล่าง

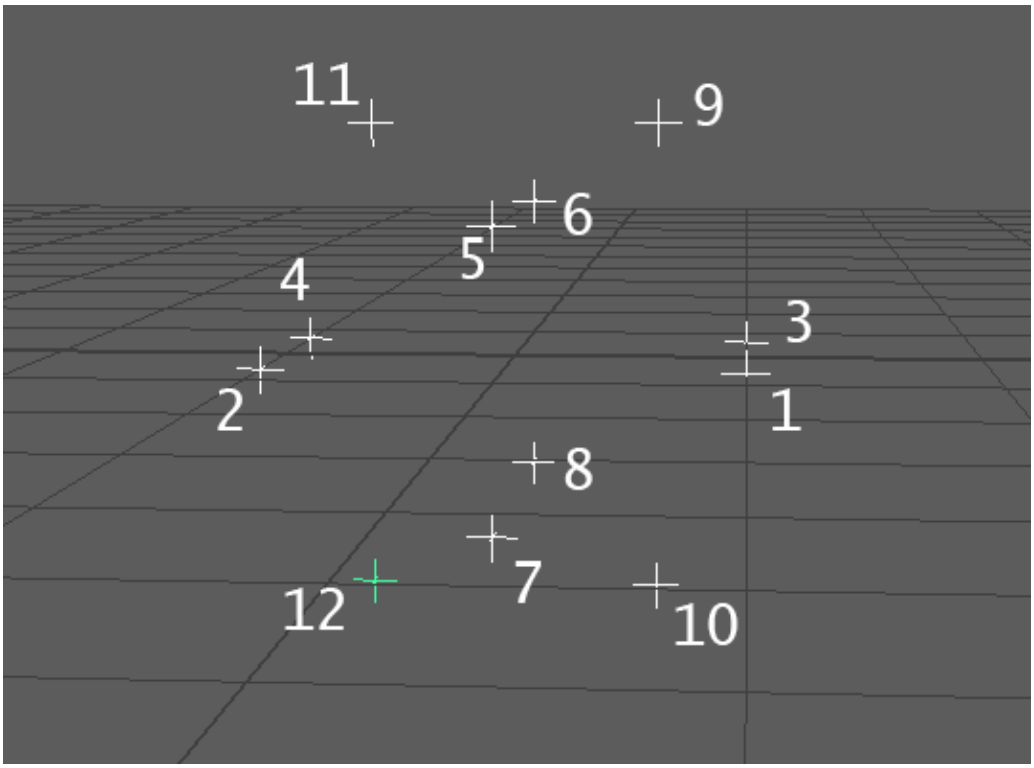

Fig 03-03: แสดง space locators ที่มีการปรับขนาดให้เหมาะสมกับการทำงาน

เมื่อเราได้ตำแหน่ง locators ครบทั้ง 12 จุดแล้ว เราจะสร้างพื้นผิว (faces) ขึ้นมาจากการเชื่อมตำแหน่งอ้างอิงเหล่านี้ โดยสามารถทำได้ด้วยคำสั่งสร้างพื้นผิว polyCreateFacet

```
polyCreateFacet
-name face01
-point ($v11.x) ($v11.y) ($v11.z)
-point ($v09.x) ($v09.y) ($v09.z)
-point ($v05.x) ($v05.y) ($v05.z);
```

จากตัวอย่าง เราสั่งสร้างพื้นผิวขึ้นมาจากการเชื่อมจุดสามจุด โดยจุดแรกอยู่ที่ตำแหน่ง X = $v11.x, Y = $v11.y และ Z = $v11.z จุดที่สองอยู่ที่ X = $v09.x, Y = $v09.y และ Z = $v09.z และจุดที่สามที่ X = $v05.x, Y = $v05.y และ Z = $v05.z ตามลำดับ เพื่อเชื่อมจุดทั้ง 12 จุด เราสามารถสร้างพื้นผิวทั้งหมด 20 พื้นผิว ซึ่ง list ของพื้นผิวที่จะสร้างมีดังนี้

951301: Advanced Animation and Rendering Techniques

Face ที่ 1 เชื่อมจุด 11, 09, 05;
Face ที่ 2 เชื่อมจุด 11, 06, 09;
Face ที่ 3 เชื่อมจุด 11, 02, 04
Face ที่ 4 เชื่อมจุด 11, 02, 05;
Face ที่ 5 เชื่อมจุด 11, 06, 04;
Face ที่ 6 เชื่อมจุด 03, 01, 09
Face ที่ 7 เชื่อมจุด 03, 06, 09;
Face ที่ 8 เชื่อมจุด 01, 05, 09;
Face ที่ 9 เชื่อมจุด 10, 03, 01
Face ที่ 10 เชื่อมจุด 10, 07, 01;
Face ที่ 11 เชื่อมจุด 05, 07, 01;
Face ที่ 12 เชื่อมจุด 05, 07, 02
Face ที่ 13 เชื่อมจุด 12, 07, 02;
Face ที่ 14 เชื่อมจุด 12, 07, 10;
Face ที่ 15 เชื่อมจุด 12, 08, 10
Face ที่ 16 เชื่อมจุด 03, 08, 10;
Face ที่ 17 เชื่อมจุด 03, 08, 06;
Face ที่ 18 เชื่อมจุด 04, 08, 06
Face ที่ 19 เชื่อมจุด 04, 08, 12;
Face ที่ 20 เชื่อมจุด 04, 02, 12

เมื่อ list จุดที่ต้องเชื่อมกันครบแล้ว เราสามารถใช้คำสั่งในการสร้างพื้นผิวตาม list ที่เราได้มาดังนี้

```
polyCreateFacet
-name face01
-point ($v11.x) ($v11.y) ($v11.z)
-point ($v09.x) ($v09.y) ($v09.z)
-point ($v05.x) ($v05.y) ($v05.z);
polyCreateFacet
-name face02
-point ($v11.x) ($v11.y) ($v11.z)
-point ($v06.x) ($v06.y) ($v06.z)
-point ($v09.x) ($v09.y) ($v09.z);
polyCreateFacet
-name face03
-point ($v02.x) ($v02.y) ($v02.z)
-point ($v04.x) ($v04.y) ($v04.z)
-point ($v11.x) ($v11.y) ($v11.z);
polyCreateFacet
-name face04
-point ($v11.x) ($v11.y) ($v11.z)
-point ($v02.x) ($v02.y) ($v02.z)
-point ($v05.x) ($v05.y) ($v05.z);
polyCreateFacet
-name face05
-point ($v04.x) ($v04.y) ($v04.z)
-point ($v11.x) ($v11.y) ($v11.z)
-point ($v06.x) ($v06.y) ($v06.z);
polyCreateFacet
-name face06
-point ($v09.x) ($v09.y) ($v09.z)
-point ($v03.x) ($v03.y) ($v03.z)
-point ($v01.x) ($v01.y) ($v01.z);
```


951301: Advanced Animation and Rendering Techniques

```
polyCreateFacet
-name face07
-point ($v09.x) ($v09.y) ($v09.z)
-point ($v03.x) ($v03.y) ($v03.z)
-point ($v06.x) ($v06.y) ($v06.z);
polyCreateFacet
-name face08
-point ($v09.x) ($v09.y) ($v09.z)
-point ($v01.x) ($v01.y) ($v01.z)
-point ($v05.x) ($v05.y) ($v05.z);
polyCreateFacet
-name face09
-point ($v10.x) ($v10.y) ($v10.z)
-point ($v03.x) ($v03.y) ($v03.z)
-point ($v01.x) ($v01.y) ($v01.z);
polyCreateFacet
-name face10
-point ($v10.x) ($v10.y) ($v10.z)
-point ($v07.x) ($v07.y) ($v07.z)
-point ($v01.x) ($v01.y) ($v01.z);
polyCreateFacet
-name face11
-point ($v05.x) ($v05.y) ($v05.z)
-point ($v07.x) ($v07.y) ($v07.z)
-point ($v01.x) ($v01.y) ($v01.z);
polyCreateFacet
-name face12
-point ($v05.x) ($v05.y) ($v05.z)
-point ($v07.x) ($v07.y) ($v07.z)
-point ($v02.x) ($v02.y) ($v02.z);
polyCreateFacet
-name face13
-point ($v12.x) ($v12.y) ($v12.z)
-point ($v07.x) ($v07.y) ($v07.z)
-point ($v02.x) ($v02.y) ($v02.z);
polyCreateFacet
-name face14
-point ($v12.x) ($v12.y) ($v12.z)
-point ($v07.x) ($v07.y) ($v07.z)
-point ($v10.x) ($v10.y) ($v10.z);
polyCreateFacet
-name face15
-point ($v12.x) ($v12.y) ($v12.z)
-point ($v08.x) ($v08.y) ($v08.z)
-point ($v10.x) ($v10.y) ($v10.z);
polyCreateFacet
-name face16
-point ($v03.x) ($v03.y) ($v03.z)
-point ($v08.x) ($v08.y) ($v08.z)
-point ($v10.x) ($v10.y) ($v10.z);
polyCreateFacet
```

```
-name face17
-point ($v03.x) ($v03.y) ($v03.z)
-point ($v08.x) ($v08.y) ($v08.z)
-point ($v06.x) ($v06.y) ($v06.z);
polyCreateFacet
-name face18
-point ($v04.x) ($v04.y) ($v04.z)
-point ($v08.x) ($v08.y) ($v08.z)
-point ($v06.x) ($v06.y) ($v06.z);
polyCreateFacet
-name face19
-point ($v04.x) ($v04.y) ($v04.z)
-point ($v08.x) ($v08.y) ($v08.z)
-point ($v12.x) ($v12.y) ($v12.z);
polyCreateFacet
-name face20
-point ($v04.x) ($v04.y) ($v04.z)
-point ($v02.x) ($v02.y) ($v02.z)
-point ($v12.x) ($v12.y) ($v12.z);
```

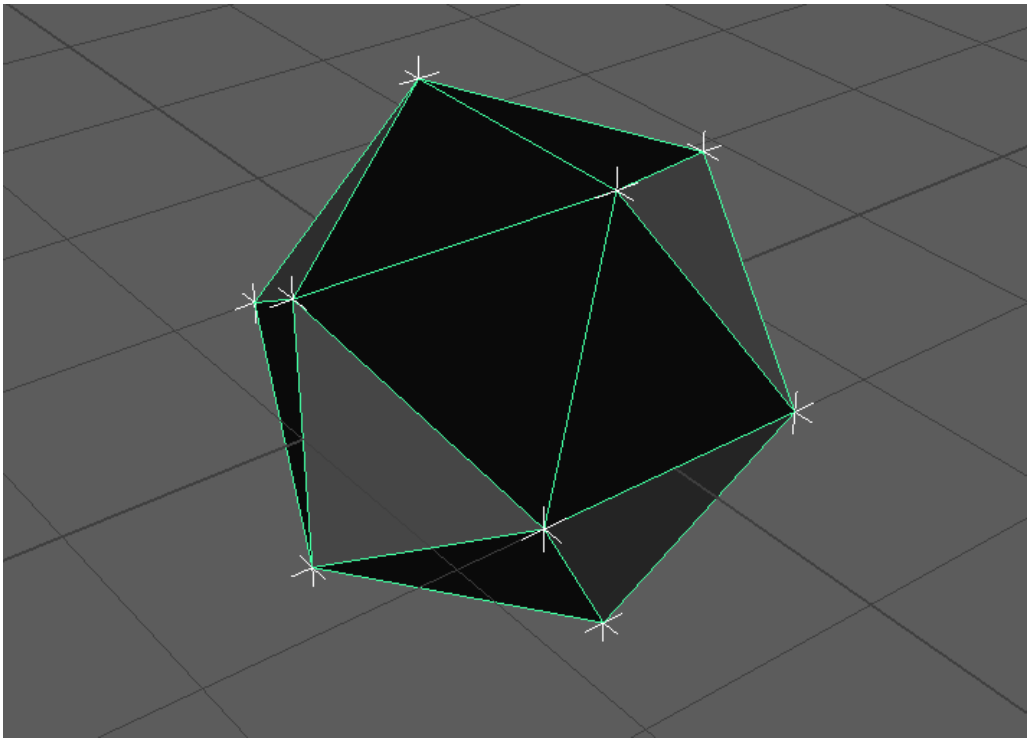

Fig 03-04: รูปทรง polyhedron ที่ได้จากการเชื่อมจุด

เมื่อ execute คำสั่งจะได้วัตถุดังภาพตัวอย่าง ถ้าเราลอง select ไปที่วัตถุ จะพบว่าพื้นผิวแต่ละอันยังแยกกันอยู่ไม่รวมเป็นหนึ่งเดียวกัน โดยมีชื่อว่า face01 – face20 ตามที่เราตั้งไว้ ขั้นตอนต่อไปเราจะรวม

951301: Advanced Animation and Rendering Techniques

พื้นผิวทั้งหมดเข้าเป็นวัตถุเดียวกันโดยตั้งชื่อว่า polyhedron โดยสามารถทำได้ด้วยคำสั่ง polyUnite ตามด้วย flag ชื่อที่เราต้องการ ตามด้วย faces ทั้งหมดที่เราต้องการรวมดังตัวอย่างด้านล่าง อย่าลืมว่าต้องเขียนชื่อ faces ให้ถูกต้อง

```
polyUnite -name polyHedron
face01 face02 face03 face04 face05 face06 face07 face08
face09 face10 face11 face12 face13 face14 face15 face16
face17 face18 face19 face20;
```

เราจะได้วัตถุแบบ polyhedron ขึ้นมา จากนั้นเราสามารถ save scripts ไปไว้บน shelf เพื่อเรียกใช้ในโอกาสต่อไปได้ ทีนี้เราก็จะมี polygon primitives เพิ่มขึ้นมาอีกหนึ่งชนิด

จากตัวอย่างนี้เราอาจพบปัญหาเรื่องของ Polygon Normal หันไปในทิศทางที่ไม่ต้องการ เนื่องจากโปรแกรมไม่เข้าใจว่าด้านไหนคือด้านนอกของวัตถุ ซึ่งเราสามารถแก้ไขได้ด้วยคำสั่ง ReversePolygonNormals โดยให้เปลี่ยนโหมดการทำงานให้เป็นแบบ face แล้ว select เลือก face ที่ต้องการกลับด้านแล้ว run คำสั่ง ReversePolygonNormals จากตัวอย่างที่เราทำนี้จะมี face ที่ 0, 1, 2, 5, 7, 9, 11, 13, 15, 17, 19 ที่กลับด้านอยู่ เราสามารถกลับด้าน faces ทั้งหมดได้ด้วยคำสั่งดังนี้

```
// flip normal
// if any face has wrong normal do this
select -r polyHedron.f[7] polyHedron.f[2] polyHedron.f[19]
polyHedron.f[11]
polyHedron.f[17] polyHedron.f[1] polyHedron.f[0] polyHedron.f[5]
polyHedron.f[9]
polyHedron.f[15] polyHedron.f[13];
ReversePolygonNormals;
```

Reference

- Wilkins, M. R. and Kazmier, C. (2005) *MEL Scripting for Maya Animations*, Morgan Kaufmann Publishers, Elsevier Inc.
- Galanakis, R. (2014) *Practical Maya Programming with Python*, Packt Publishing Ltd.
- Stripinis, D. (2003) *The MEL Companion: Maya Scripting 3D Artists*, Charles River Media, INC

951301: Advanced Animation and Rendering Techniques