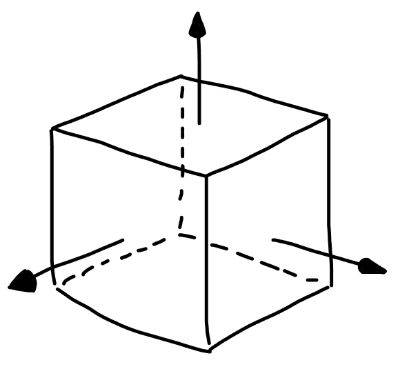
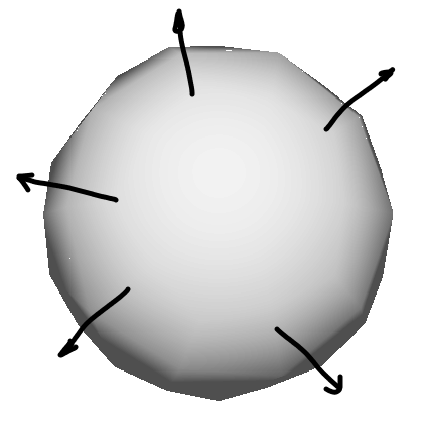
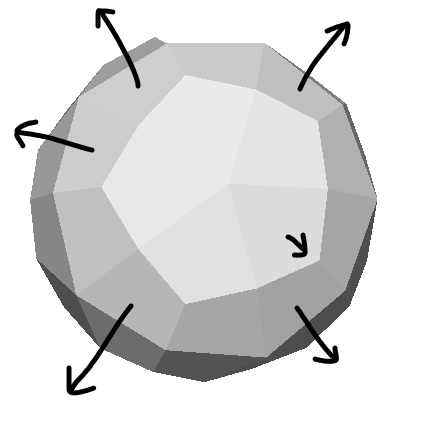
0７ – FBX③

**立方体の法線**

　法線のデータは実際のデータと、インデックスという形で分かれている。法線にもインデックスが利用される理由は立方体の法線について考えるとわかりやすい。法線は面の向きを表す情報だが、実際は頂点に保存されている。各頂点の法線を補間することで面の法線を表現している。そのため、立方体のある面の法線はある一方向を指していることから、面を構成するすべての頂点の法線はすべて等しいことがわかる。

　法線のデータを頂点ごとに保存するのは同じデータが複数あることになり、無駄になってしまう。そのため、法線でもインデックスが用いられる。これは後々出てくるテクスチャ座標にも同じことが言える。

**球の法線**



　立方体は面ごとに法線が必要だが、球に関してはその限りでない。もし、球の法線が面ごとについている場合、光の反射から角張った球として見えてしまう。面ではなく頂点に法線がついている場合、頂点同士の法線を補間するため、少ない頂点数でもなめらかな球に見える。

　FBXではこれらの法線の違いを指定して保存している。

-- 法線の読み込み(FBXParser.cpp) --

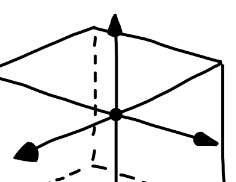
|  |
| --- |
| ptr = FindKeyword(pGeometry, "LayerElementNormal:");  char\* pNormal = ptr; // LayerElementNormalの位置を一旦保存  if (ptr != NULL)　{  // マッピングの読み取り  ptr = FindKeyword(ptr, "MappingInformationType:");  if (ptr != NULL)　{  ptr = FindKeyword(ptr, "\"");  if (ptr != NULL)　{  ptr += 1;  if (ptr == strstr(ptr, "ByPolygonVertex")) {  m\_normalMapping = BY\_POLYGON\_VERTEX;  } else if (ptr == strstr(ptr, "ByControlPoint")) {  m\_normalMapping = BY\_CONTROL\_POINT;  } else {  m\_normalMapping = ALL\_SAME;  }  }  }　else  m\_normalMapping = ALL\_SAME;  // リファレンスの読み取り  ptr = FindKeyword(ptr, "ReferenceInformationType:");  if (ptr != NULL) {  ptr = FindKeyword(ptr, "\"");  if (ptr != NULL)　{  ptr += 1;  if (ptr == strstr(ptr, "Direct"))　{  m\_normalReference = DIRECT;  }　else if (ptr == strstr(ptr, "IndexToDirect"))　{  m\_normalReference = INDEX\_TO\_DIRECT;  }  }  } else  m\_normalReference = DIRECT; |

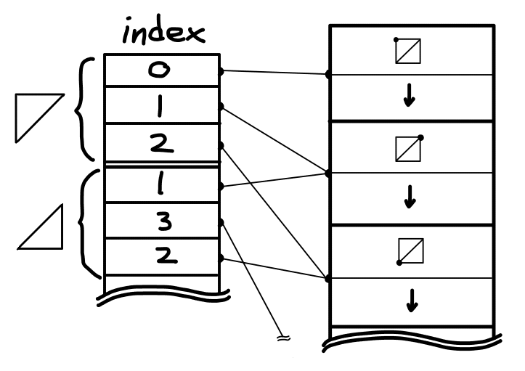
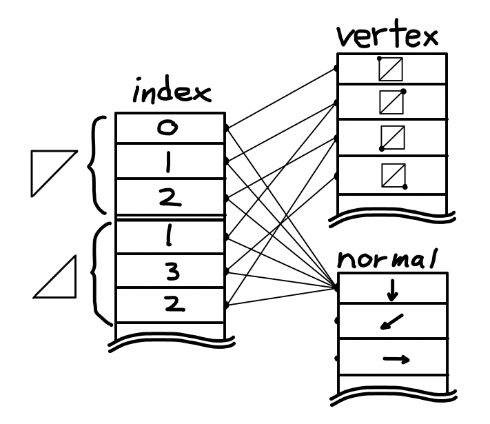
|  |
| --- |
| // 法線データの読み取り  ptr = FindKeyword(ptr, "Normals:");  if (ptr != NULL) {  ptr = FindKeyword(ptr, "\*");  if (ptr != NULL) {  int num = atoi(ptr + 1);  m\_normal.resize(num / 3);  ptr = FindKeyword(ptr, "a:");  if (ptr != NULL) {  ptr += 2;  for (int i = 0; i < m\_normal.size(); ++i) {  m\_normal[i].x = atof(ptr);  ptr = FindKeyword(ptr, ",") + 1;  m\_normal[i].y = atof(ptr);  ptr = FindKeyword(ptr, ",") + 1;  m\_normal[i].z = atof(ptr);  ptr = FindKeyword(ptr, ",") + 1;  }  }  }  }  // 法線インデックスの読み取り  ptr = FindKeyword(pNormal, "NormalsW:");  if (ptr != NULL) {  ptr = FindKeyword(ptr, "\*");  if (ptr != NULL) {  int num = atoi(ptr + 1);  ptr = FindKeyword(ptr, "a:");  if (ptr != NULL) {  ptr += 2;  for (int i = 0; i < num; ++i) {  m\_normalIndex.push\_back(atoi(ptr));  ptr = FindKeyword(ptr, ",") + 1;  }  }  }  }  } |

-- デバッグ表示(FBXParser.cpp) --

|  |
| --- |
| void FBXParser::DebugDrawNormal()　{  bool isMappingPolygon = m\_normalMapping == BY\_POLYGON\_VERTEX;  int normNum = isMappingPolygon ? m\_idx.size() : m\_vtx.size();  glBegin(GL\_LINES);  for (int i = 0; i < normNum; ++i) {  int idx = isMappingPolygon ? m\_idx[i] : i;  glVertex3f(m\_vtx[idx].x, m\_vtx[idx].y, m\_vtx[idx].z);  glVertex3f(  m\_vtx[idx].x + m\_normal[i].x,  m\_vtx[idx].y + m\_normal[i].y,  m\_vtx[idx].z + m\_normal[i].z);  }  glEnd();  } |

**法線情報**

　少ないメモリで保存するために、インデックスを用いて効率的に保存している。しかし、描画する際にはメモリ上に巨大なデータとして展開されるという点は注意したい。例えば立方体のある頂点について考えたとき、座標はデータ１つ分だが、その頂点に含まれる法線は3つのため、最終的には「　　　」の頂点を作らないといけない。テクスチャ座標なども含めると組み合わせはより多くなる。



　左図はFBXで保存されている状態を、右図は実際に描画する際の状態を図示している。OpenGLならglVertex3fやglNormal3f関数で左図のままでも描画できるが、DirectXや配列でまとめて描画する際は右図の状態でないといけない。

--　法線のインデックス展開対応(FBXParser.cpp) --

|  |
| --- |
| **IdxList idxWork;**  ptr = FindKeyword(pGeometry, "PolygonVertexIndex:");  if (ptr != NULL)　{  ptr = FindKeyword(ptr, "\*");  if (ptr != NULL)　{  int num = atoi(ptr + 1);  ptr = FindKeyword(ptr, "a:");  if (ptr != NULL)　{  ptr += 2;  for (int i = 0; i < num; ++i)　{  **idxWork.push\_back(atoi(ptr));**  ptr = FindKeyword(ptr, ",") + 1;  }  }  }  } |

-- 法線の展開対応(FBXParser.cpp) --

|  |
| --- |
| **NormalList workNormal;**  ptr = FindKeyword(pGeometry, "LayerElementNormal:");  ～～中略～～  ptr = FindKeyword(ptr, "a:");  if (ptr != NULL)　{  ptr += 2;  for (int i = 0; i < workNormal.size(); ++i)　{  **workNormal**[i].x = atof(ptr); ptr = FindKeyword(ptr, ",") + 1;  **workNormal** [i].y = atof(ptr); ptr = FindKeyword(ptr, ",") + 1;  **workNormal** [i].z = atof(ptr); ptr = FindKeyword(ptr, ",") + 1;  }  } |

-- ポリゴンと法線のインデックスを同時展開(FBXParser.cpp) --

|  |
| --- |
| bool isNormalMappingPolygon = m\_normalMapping == BY\_POLYGON\_VERTEX;  int idxCnt = 0;  for (int i = 0; i < idxWork.size(); ++i)　{  int idx = idxWork[i];  ++idxCnt;  if (idxCnt >= 4)　{  m\_idx.push\_back(m\_idx[m\_idx.size() - 3]);  m\_idx.push\_back(m\_idx[m\_idx.size() - 2]);  if (isNormalMappingPolygon)　{  m\_normal.push\_back(m\_normal[m\_normal.size() - 3]);  m\_normal.push\_back(m\_normal[m\_normal.size() - 2]);  }  } |

|  |
| --- |
| if (idx < 0)　{  idx = ~idx;  idxCnt = 0;  }  m\_idx.push\_back(idx);  if (isNormalMappingPolygon)  m\_normal.push\_back(workNormal[i]);  }  if (!isNormalMappingPolygon)  m\_normal = workNormal; |

-- 法線情報の取得(FBXParser.h) --

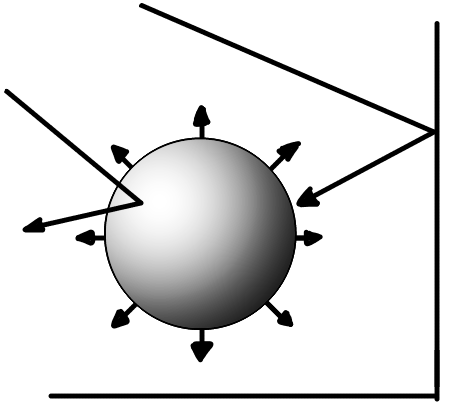
|  |
| --- |
| bool IsNormalMappingPolygon() { return m\_normalMapping == BY\_POLYGON\_VERTEX; }  bool IsNormalReferenceIndex() { return m\_normalReference == INDEX\_TO\_DIRECT; }  Normal\* GetNormal() { return m\_normal.data(); }  unsigned long\* GetNormalIndex() { return m\_normalIndex.data(); } |

-- 法線に対応した描画処理(Startup.cpp) --

|  |
| --- |
| FBXParser::Vertex\* pVtx = g\_pFBX->GetVertex();  unsigned long\* pIdx = g\_pFBX->GetIndex();  FBXParser::Normal\* pNormal = g\_pFBX->GetNormal();  unsigned long\* pNrmIdx = g\_pFBX->GetNormalIndex();  glBegin(GL\_TRIANGLES);  for (int i = 0; i < g\_pFBX->GetIndexNum(); ++i)　{  int idx = pIdx[i];  int nrmIdx = g\_pFBX->IsNormalMappingPolygon() ? i : idx;  if (g\_pFBX->IsNormalReferenceIndex()) { nrmIdx = pNrmIdx[nrmIdx]; }  glNormal3f(pNormal[nrmIdx].x, pNormal[nrmIdx].y, pNormal[nrmIdx].z);  glVertex3f(pVtx[idx].x, pVtx[idx].y, pVtx[idx].z);  }  glEnd(); |

**ライト設定**

　OpenGLでは関数一つでライトを設定することが出来る。具体的にどのように色がつくのかは内部的に計算されており、ライトの色や、「　　材質(マテリアル)　　」にもとづいて色がつく。

　設定そのものは最初から無効になっており、ライトの計算を行いたいときだけ有効にする。また、複数の光源を設定できるが、光源ごとに設定を行う必要がある。

　代表的なマテリアルは、

物体の表面で反射する光を表す「　　　拡散光(Diffuse)　　　」

周辺で反射した光を表す「　　　環境光(Ambient)　　　」

物体の金属特性を表す「　　　鏡面反射光(Specular)　　　」がある。

-- ライトそのものの有効化(Startup.cpp) --

|  |
| --- |
| glEnable(GL\_LIGHTING);  glEnable(GL\_LIGHT0); |

-- ライトを適用したいオブジェクトのみにライトを有効化(Startup.cpp) --

|  |
| --- |
| glEnable(GL\_LIGHTING);  ～～描画処理～～  glDisable(GL\_LIGHTING); |

-- ライトの向きを変える(Startup.cpp) --

|  |
| --- |
| float dir[] = { 1.0f, -1.0f, -1.0f, 0.0f };  glLightfv(GL\_LIGHT0, GL\_POSITION, dir); |

-- スペキュラー(鏡面反射光)の設定(Startup.cpp) --

|  |
| --- |
| float specular[] = { 1,1,1,1 };  glMaterialfv(GL\_FRONT, GL\_SPECULAR, specular);  glMaterialf(GL\_FRONT, GL\_SHININESS, 128.0f); |