MÜHENDİSLİK FAKÜLTESİ
BİTİRME PROJESİ

Akın DEMİRTUĞ
140205054

İÇİNDEKİLER

1. GİRİŞ

2. PROJE

3. SONUÇ

4. KAYNAKÇA

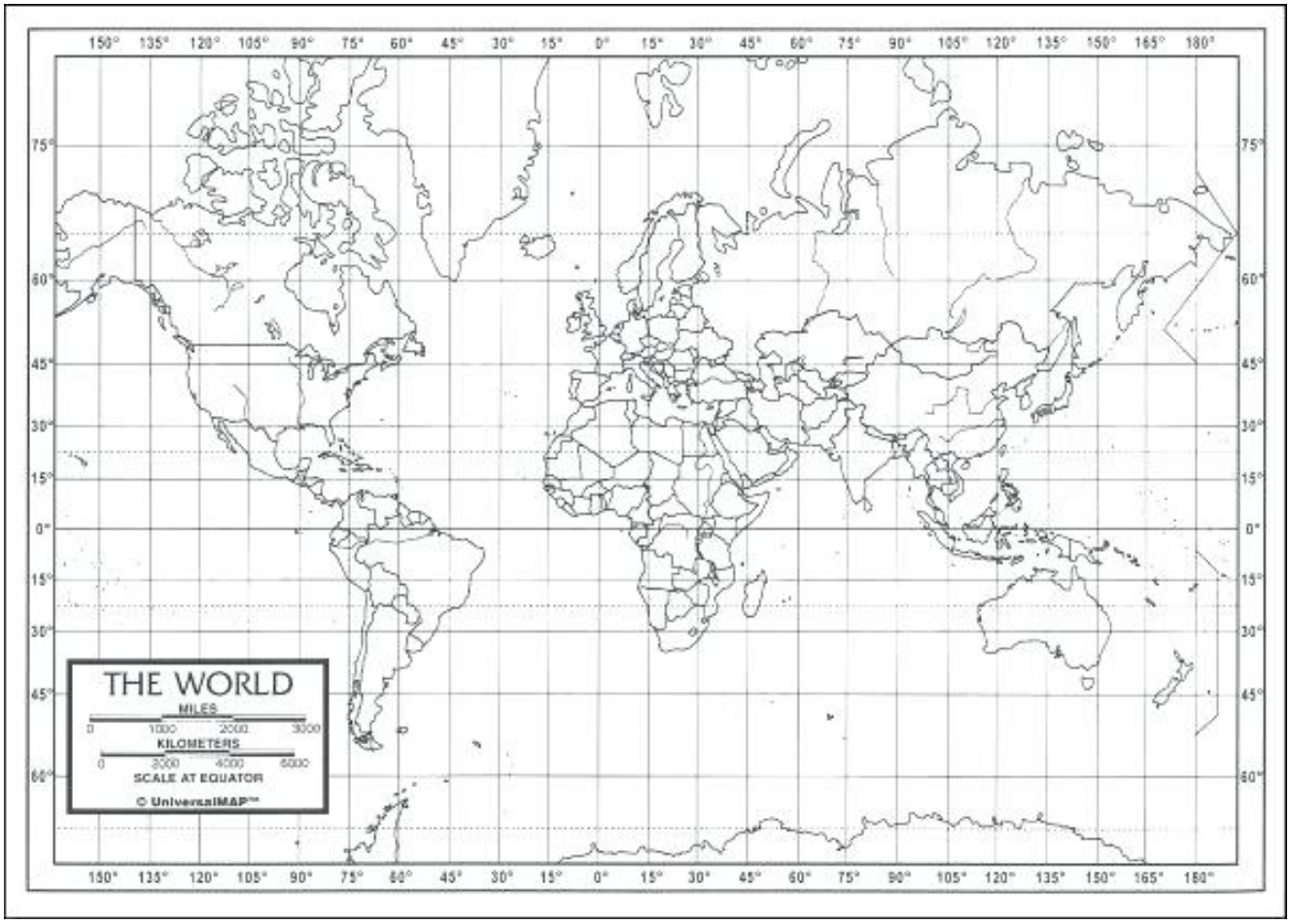
1. GİRİŞ

Bu proje kapsamında daha önceden yapmış olduğumuz grafik motor üzerine bir sayısal dünya tasarımı yapacağız.

2. PROJE

Sistemin oluşabilmesi için öncelikle bir 3 boyutlu kürenin oluşması gerekmektedir. Dünya bilindiği üzere tepesinden 1/297 oranında basık olduğundan matematiksel olarak Spheroid sınıfındadır. Bir spheroidin ekrana çizilmesi ile ilgili bir çizim komutu bulunmamaktadır bu yüzden bir spheroid yaklaşımını biz kendimiz yapacağız.

Önceliki olarak bilgisayardaki bütün 3 boyutlu çizimlerin üçgenlerden oluştuğunu hatırlayalım. Eğer bir şekil çizmek istiyorsak bunu alt üçgenlerine bölmeliyiz. Çizimlerimizi yaparken bize kolaylık sağlaması amacı ile çizimi bir 2 boyutlu harita olarak hazırlayıp sonra bunu 3 boyutlu bir cisme çevireceğiz.



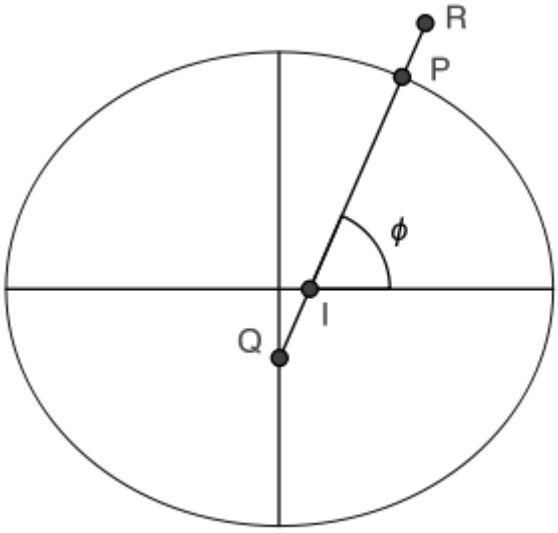
Yukarıda verilen dünya haritasına göz gezdirdiğimizde enlemler ve boylamların kesişerek dikdörtgenler oluşturduğuna ve herbir dikdörtgenin sağ üst köşesinden solt alt köşesine çizilecek bir çizgiyle iki üçgen şeklinde ifade edilebileceğine dikkat edelim. Bu aşamada yapılması gereken ise enlem-boylam ikililerinin (x,y,z) koordinatlarına çevrilmeleridir. Bir spheroid üzerinde bu işin aşağıdaki formüllerle yapılabilmesi mümkündür.

X = (N(enlem) + h) * cos(enlem) * cos(boylam)

Y = (N(enlem) + h) * cos(enlem) * sin(boylam)

Z = (b*b / a*a * N(enlem) + h) sin(enlem)

N(enlem) = a*a/(sqrt(a*acos(enlem)*cos(enlem) + b*b*sin(enlem)*sin(enlem))



Yukarıdaki formülizasyonu kullanarak bir kod yazalım.

```
float N(float phi, float a, float b)
{
	return (a*a) / sqrt((a*a)*cos(phi)*cos(phi) + (b*b)*sin(phi)*sin(phi));
}

int main()
{
	qball_camera *cam = new qball_camera();

	eng.cam = cam;
	eng.init(1024, 768);
	eng.maxfps = 25;

	float r = 6.371f;
	float a = 6.3567523f;
	float b = 6.3781370f;
	float h = 0.0f;
	float num_lat = 9;
	float num_long = 18;

	glm::vec3** globe = new glm::vec3*[num_lat + 1];
	for (int i = 0; i < num_lat + 1; ++i)
		globe[i] = new glm::vec3[num_long + 1];

	pointcloud pc;
	for (size_t i = 0; i < num_lat + 1; i++)
	{
```

```
			float lat = ((glm::pi<float>() / num_lat)*i) + glm::pi<float>() / 2;

			for (size_t j = 0; j<num_long + 1; j++)
			{
				float lon = ((2 * glm::pi<float>() / num_long)*j);


				float x = (N(lat, a, b) + h) * cos(lat) * cos(lon);

				float y = (N(lat, a, b) + h) * cos(lat) * sin(lon);

				float z = (((b*b) / (a*a)) * N(lon, a, b) + h) * sin(lat);

				glm::vec3 pt = { x, y, z };
				globe[i][j] = pt;
				pc.addpoint(pt);
			}
		}

		pc.init();
		pc.position = { 0.0, 0.0, 0.0 };
		eng.sc->meshes.push_back(&pc);

		eng.sc->generate_shaders();
		eng.run();

		glfwTerminate();
		return 0;
}
```

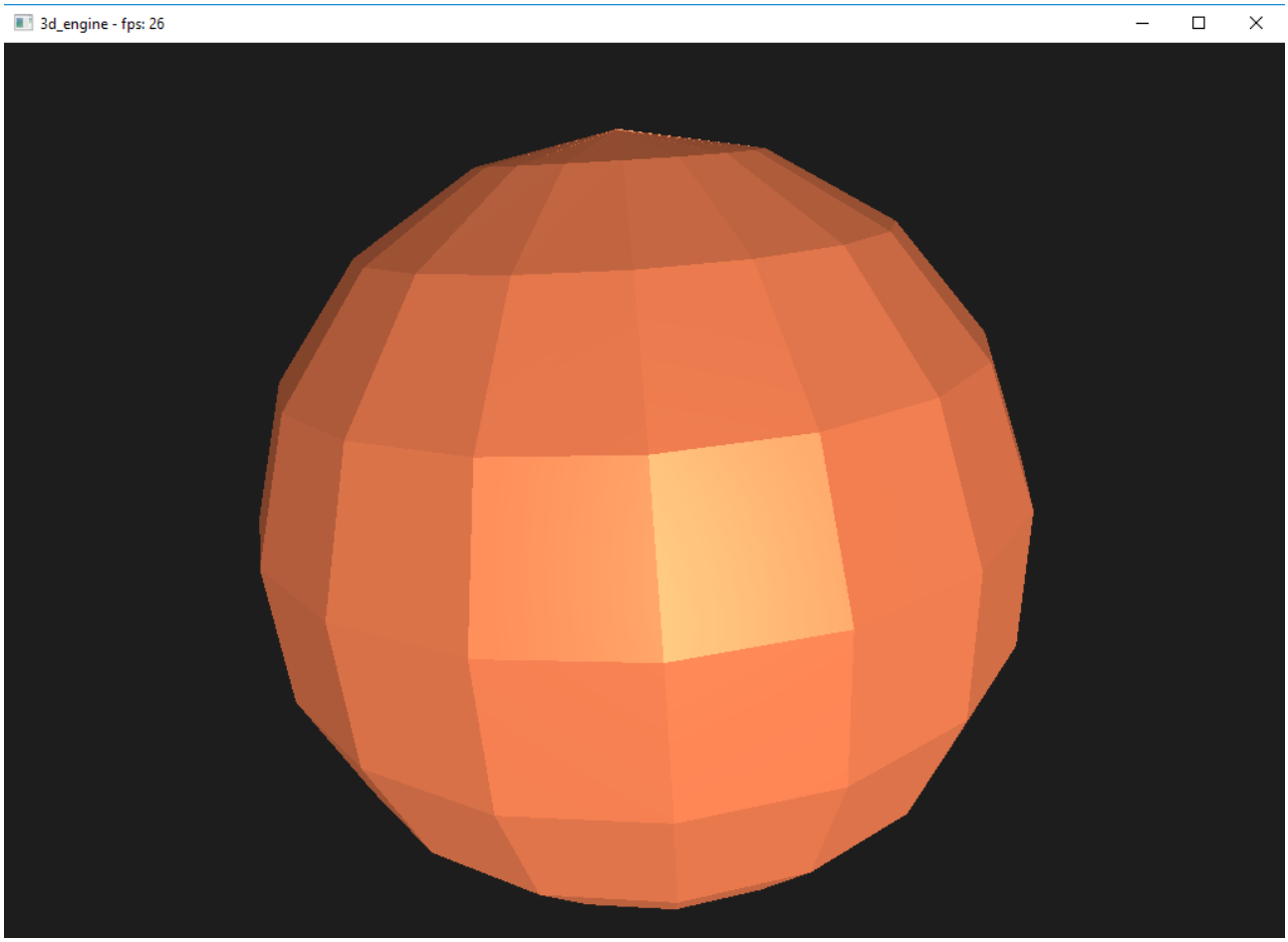
Bu kodumuzu sistem üzerinde çalıştırdığımızda aşağıdaki sonucu elde etmemiz mümkündür.

Elde edilen noktaların doğru bir biçimde sisteme sıra ile verilerek üçgenlerin oluşturulmasıda yine mümkündür. Şimdi kodumuzda bir ufak değişiklikle bunuda sağlayalım.

```
float N(float phi, float a, float b)
{
	return (a*a) / sqrt((a*a)*cos(phi)*cos(phi) + (b*b)*sin(phi)*sin(phi));
}

int main()
{
	qball_camera *cam = new qball_camera();

	eng.cam = cam;
	eng.init(1024, 768);
	eng.maxfps = 25;

	float r = 6.371f;
	float a = 6.3567523f;
	float b = 6.3781370f;
	float h = 0.0f;
	float num_lat = 9;
	float num_long = 18;


	glm::vec3** globe = new glm::vec3*[num_lat + 1];
	for (int i = 0; i < num_lat + 1; ++i)
		globe[i] = new glm::vec3[num_long + 1];


	vector<glm::vec3> vertices;
	vector<glm::vec3> normals;
	for (size_t i = 0; i < num_lat + 1; i++)
	{
		float lat = ((glm::pi<float>() / num_lat)*i) + glm::pi<float>() / 2;

		for (size_t j = 0; j<num_long + 1; j++)
		{
			float lon = ((2 * glm::pi<float>() / num_long)*j);

			float x = (N(lat, a, b) + h) * cos(lat) * cos(lon);

			float y = (N(lat, a, b) + h) * cos(lat) * sin(lon);

			float z = (((b*b) / (a*a)) * N(lon, a, b) + h) * sin(lat);

			glm::vec3 pt = { x, y, z };
			globe[i][j] = pt;
		}
	}

	for (size_t i = 0; i < num_lat; i++)
	{
		for (size_t j = 0; j< num_long; j++)
		{
			glm::vec3 pt1 = globe[i][j];
			glm::vec3 pt2 = globe[i+1][j];
			glm::vec3 pt3 = globe[i][j+1];

			vertices.push_back(pt1);
			vertices.push_back(pt2);
			vertices.push_back(pt3);
```

```
                normals.push_back(calc_normal(pt1, pt2, pt3));
                normals.push_back(calc_normal(pt2, pt3, pt1));
                normals.push_back(calc_normal(pt3, pt1, pt2));

                glm::vec3 pt4 = globe[i+1][j];
                glm::vec3 pt5 = globe[i + 1][j + 1];
                glm::vec3 pt6 = globe[i][j + 1];

                vertices.push_back(pt4);
                vertices.push_back(pt5);
                vertices.push_back(pt6);

                normals.push_back(calc_normal(pt4, pt5, pt6));
                normals.push_back(calc_normal(pt5, pt6, pt4));
                normals.push_back(calc_normal(pt6, pt4, pt5));
            }
        }
        colormesh cm(vertices, normals);
        cm.position = { 0.0, 0.0, 0.0 };
        eng.sc->meshes.push_back(&cm);

        eng.sc->generate_shaders();
        eng.run();

        glfwTerminate();
        return 0;
}
```

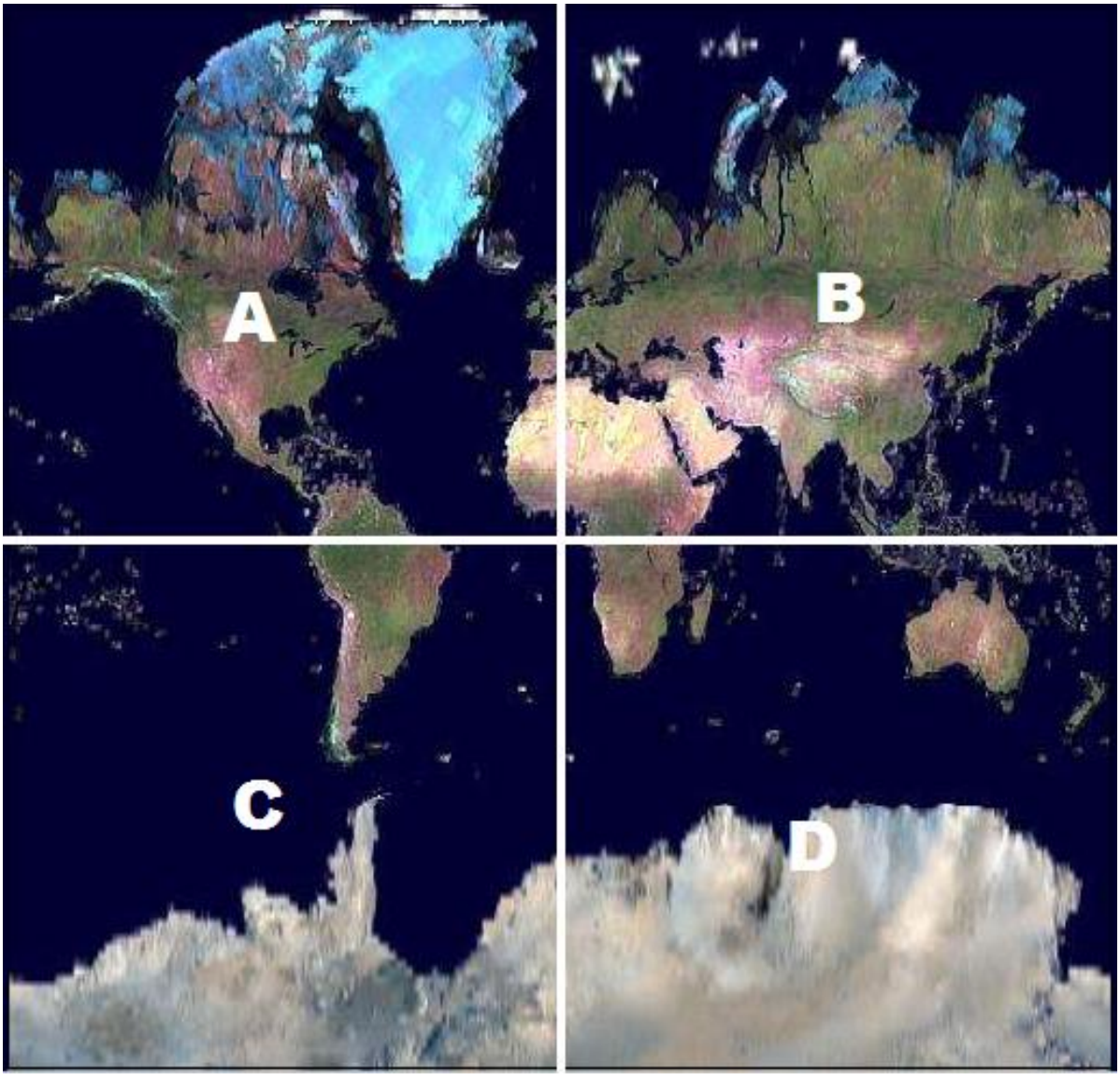
Sonuç olarak elde edeceğimiz ise aşağıdaki gibidir.



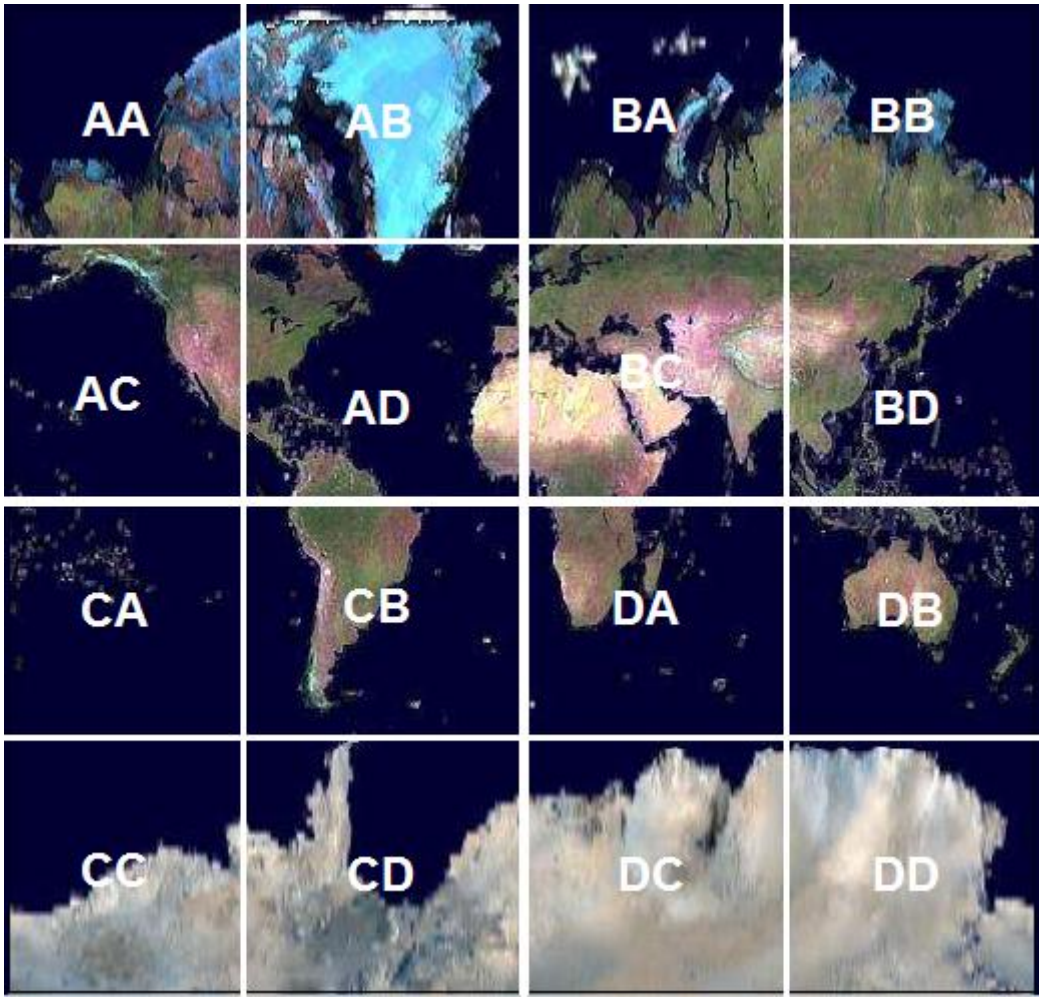
Şeklin köşeli olma gerekçesi kullandığımız üçgen sayısı ile ilgilidir daha fazla üçgen ile sistemi spheroid e yaklaştırmak mümkündür. Fakat daha fazla üçgenin sistemi daha fazla zorlayacağı unutulmamalıdır.

Bu yaklaşım yöntemimiz ile her ne kadar iyi bir sonuç elde etsekte bu modelin kullanılması mümkün değildir. Bunun başlıca gerekçesi 3 boyutlu programlamada bir cismin üzerine sadece bir resim yapıştırılabilmesidir. Biz yaptığımız sistemde matematiksel olarak yaklaşılan bölge üzerinde daha detaylı haritalar göstermek isteyeceğiz bu haritalara ait bilgiyi dünya şekli üzerine eklememiz mümkün olmayacağından dünyanın yekpare değil parçalar halinde oluşturulması gerekmektedir.

Birçok harita sistemi verilerin depolanmasında ve gösteriminde quadtree sistemini kullanmaktadır. Her ne kadar De Juri olmasa da De Facto bir standart olarak karşımıza çıktığından bizde projemizde bu yolu takip edeceğiz şimdi kısaca quadtree gösterim sistemini aşağıdaki resimde inceleyelim.



Yukarıda da görüldüğü üzere bir quadtree temsilinde dünya zoom seviyesine göre parçalara bölünmektedir. En uzaktan görüntüyü temsil eden birinci seviyede dünya enine ve boyuna 4^zoom seviyesi kadar yani 4 parçaya ayrılmaktadır ve her bir parça 256x256 piksel resimle temsil edilmektedir. Bu parçalanmada herbir parça A/B/C/D olarak isimlendirilmektedir. 2. zoom seviyesinde ise dünyanın her bir parçası da kendi içerisinde yine 4 alt parçaya bölünmektedir.



Bu sistem tasarımsal olarak tam ve dolu 4 lü ağaç yapısıdır. Sistemimiz üzerinde biz bunu quadtile sınıfı ile temsil etmekteyiz. Şimdi basit olarak kodlarına göz atalım.

```
class quadtile
{
public:
	quadtile* children;
	string quadkey;
	vector<glm::vec3> vertices;
	vector<glm::vec3> normals;
	vector<glm::vec2> uvs;

	bool requested = false;

	shared_ptr<texturemesh> tm;
	wstring fname;
	int platebase;
	double lat_center = 0;
```

```
        double lon_center = 0;


        quadtile();
        ~quadtile();
        void initchildren(string _quadkey = "");

        quadtile* getchild(char c);
        quadtile* gettile(string tile);
        void germinate(string tile);

        void prune(string root, set<string>& sprouts, int zoomlevel);
        vector<quadtile*> calculatesubtiles(glm::vec3 cameraPos, int zoomlevel);
        void invalidate(string tile);
        void buildplates();

};
```

Görüldüğü gibi bütün düğümlerin kendi içerisinde 4 alt düğümü bulunmaktadır. Bu düğümler buildplates fonksiyonu aracılığıyla 3 boyutlu olarak hesaplanmaktadır. Hesaplama sırasında ayrıca UV haritalamalarıda yapılmaktadır.

```
void quadtile::buildplates()
{
        if (quadkey.size() < 1)
                return;

        int platenum = platebase - quadkey.size() > 4 ? platebase - quadkey.size() : 4;

        double circumference = 2 * glm::pi<double>() * 6378137.0f;
        double mapsize = pow(2, quadkey.size()) * 256;

        double x1 = 0;
        double x2 = mapsize;
        double y1 = 0;
        double y2 = mapsize;


        //2 üzeri seviye kadar en boy parça
        for (size_t i = 0; i < quadkey.size(); i++)
        {
                if (quadkey[i] == 'A')
                {
                        y2 = (y2 + y1) / 2;
                        x2 = (x2 + x1) / 2;
                }
                else if (quadkey[i] == 'B')
                {
                        x1 = (x2 + x1) / 2;
                        y2 = (y2 + y1) / 2;
                }
                else if (quadkey[i] == 'C')
                {
                        x2 = (x2 + x1) / 2;
                        y1 = (y2 + y1) / 2;
                }
                else if (quadkey[i] == 'D')
                {
                        x1 = (x2 + x1) / 2;
                        y1 = (y2 + y1) / 2;
                }
```

```
}

double centerx = (x2 + x1) / 2;
double centery = (y2 + y1) / 2;

lat_center = ytolat(circumference / mapsize * (mapsize / 2 - centery));
lon_center = (360.0 / mapsize * centerx) - 180;


double b = 6356752.3f;
double a = 6378137.0f;
double e2 = 1 - ((b*b) / (a*a));
double e = sqrt(e2);

double xstep = (x2 - x1) / platenum;
double ystep = (y2 - y1) / platenum;

int px = -1;
int py = -1;


for (size_t x = 0; x < platenum; x++)
{
	for (size_t y = 0; y < platenum; y++)
	{
		double mercx1 = x1 + x * xstep;
		double mercx2 = x1 + (x + 1) * xstep;

		double mercy1 = y1 + y * ystep;
		double mercy2 = y1 + (y + 1) * ystep;

		//2d to lat long
		double lat1 = ytolat(circumference / mapsize * (mapsize / 2 - mercy1));
		double lon1 = (360.0 / mapsize * mercx1) - 180;

		glm::vec3 ecef = lla2ecef(lat1, lon1);
		glm::vec3 topleft = { ecef.x, ecef.y, ecef.z };

		double lat2 = ytolat(circumference / mapsize * (mapsize / 2 - mercy1));
		double lon2 = (360.0 / mapsize * mercx2) - 180;

		ecef = lla2ecef(lat2, lon2);
		glm::vec3 topright = { ecef.x, ecef.y, ecef.z };

		double lat3 = ytolat(circumference / mapsize * (mapsize / 2 - mercy2));
		double lon3 = (360.0 / mapsize * mercx1) - 180;

		ecef = lla2ecef(lat3, lon3);
		glm::vec3 bottomleft = { ecef.x, ecef.y, ecef.z };

		double lat4 = ytolat(circumference / mapsize * (mapsize / 2 - mercy2));
		double lon4 = (360.0 / mapsize * mercx2) - 180;

		ecef = lla2ecef(lat4, lon4);
		glm::vec3 bottomright = { ecef.x, ecef.y, ecef.z };

		vertices.push_back(topleft);
		vertices.push_back(bottomleft);
		vertices.push_back(topright);


		glm::vec2 uvtopleft = { (x * xstep) / 256.0,  1.0 -(y * ystep) /
		256.0 };
		glm::vec2 uvbottomleft = { (x * xstep) / 256.0,  1.0 - (y + 1) *
		ystep / 256.0 };
```

```
                    glm::vec2 uvtopright = { (x + 1) * xstep / 256.0, 1.0 - (y * ystep) /
                    256.0 };

                    uvs.push_back(uvtopleft);
                    uvs.push_back(uvbottomleft);
                    uvs.push_back(uvtopright);

                    normals.push_back(calc_normal(topleft, bottomleft, topright));//topleft
                    normals.push_back(calc_normal(bottomleft, topright,
                    topleft));//bottomleft
                    normals.push_back(calc_normal(topright, topleft,
                    bottomleft));//topright

                    vertices.push_back(bottomleft);
                    vertices.push_back(bottomright);
                    vertices.push_back(topright);

                    glm::vec2 uvbottomright = { (x + 1) * xstep / 256.0, 1.0 - (y + 1) *
                    ystep / 256.0 };

                    uvs.push_back(uvbottomleft);
                    uvs.push_back(uvbottomright);
                    uvs.push_back(uvtopright);

                    normals.push_back(calc_normal(bottomleft, bottomright,
                    topright));//bottomleft
                    normals.push_back(calc_normal(bottomright, topright,
                    bottomleft));//bottomright
                    normals.push_back(calc_normal(topright, bottomleft,
                    bottomright));//topright
            }
        }

}
```

Sistemimizde bu plakalara ait resimler mapquest firmasınından online olarak temin edilmektedir. Edinilen resimler png formatındadır fakat ekran kartları raw halde dosya istediklerinden bu dosyaların çevrimleri gereklidir. Png formatı kendi içerisinde huffman ağacı yolu ile sıkıştırma yapmaktadır. Biz çevrim için Windows GDI+ ile birlikte gelen fonksiyonları kullandık.

```
        ////haritaları yükle
        string req = "https://www.mapquestapi.com/staticmap/v5/map?
        key=tGyflWkBSo8sNTv0PsRPD1F0OAxX11Yh&format=png&center=" +
               to_string(lat_center) + "," + to_string(lon_center) +
               "&size=256,256&zoom=" + to_string(quadkey.size());



        fname = L"C:\\mapdata\\" + wstring(quadkey.begin(), quadkey.end()) + L".bmp";

        if (!std::filesystem::exists("C:\\mapdata\\" + quadkey + ".bmp"))
        {
               cout << "requested: " << quadkey << endl;
               http_client hc;
               vector<unsigned char> png = hc.get_binary_page(req);
               GdiplusStartupInput gdiplusStartupInput;
               ULONG_PTR gdiplusToken;
               GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);
               CLSID   encoderClsid;
               Status  stat;
```

```
            HGLOBAL hMem = GlobalAlloc(GMEM_MOVEABLE, png.size());

            PVOID pMem = GlobalLock(hMem);
            RtlMoveMemory(pMem, &png[0], png.size());
            IStream *pStream = 0;
            HRESULT hr = CreateStreamOnHGlobal(hMem, TRUE, &pStream);

            Image* img = new Gdiplus::Image(pStream);
            GetEncoderClsid(L"image/bmp", &encoderClsid);

            stat = img->Save(fname.c_str(), &encoderClsid, NULL);
            delete img;
            GdiplusShutdown(gdiplusToken);
      }
      else
      {
            cout << "requestedD: " << quadkey << endl;
      }
```

Resimler edinildikten sonra ekran kartlarına yükleme işlemlerinin yapılması gerekmektedir. Sistemde bu resimler arka planda thread ler aracılığıyla yapılmaktadır. Fakat hızlıca yapılacak ard arda zoomlar sisteme çok kısa sürede çok fazla resim yükleme talebi göndermektedir. Sistemin akıcılığını sağlayıp daha iyi bir kullanıcı deneyimi sunabilmek adına bir thread pool sistemi tasarlanmıştır.

```
threadpool::threadpool() : maxallowed(5), running(true)
{
}
threadpool::~threadpool()
{
      stop();
}

void threadpool::queue(shared_ptr<iwork> work)
{
      unique_lock<std::mutex> lk_dead(mtx_dthreads);
      deadthreads.erase(deadthreads.begin(), deadthreads.end());
      lk_dead.unlock();

      unique_lock ul_works(mtx_works);
      works.push_back(work);
      ul_works.unlock();


      unique_lock ul_threads(mtx_threads);
      if (threads.size() < maxallowed)
      {
            shared_ptr<safethread> st(new safethread(std::bind(&threadpool::handleworks,
            this)));
            threads.push_back(st);
      }
      ul_threads.unlock();
}

void threadpool::handleworks()
{
      while (running)
      {
            shared_ptr<iwork> work;
            unique_lock<std::mutex> lk_works(mtx_works);
            if (works.size() > 0)
```

```
		{
			work = works[0];
			works.erase(works.begin());
		}
		lk_works.unlock();

		if (work.get() != NULL)
		{
			try
			{
				work->perform();
			}
			catch (...)
			{
			}
		}
		else
		{
			shared_ptr<safethread> th;

			unique_lock<std::mutex> lk_threads(mtx_threads);
			for (size_t i = 0; i < threads.size(); i++)
			{
				if (threads[i]->th->get_id() == std::this_thread::get_id())
				{
					th = threads[i];
					threads.erase(threads.begin() + i);
				}
			}
			lk_threads.unlock();

			if (th != nullptr)
			{
				unique_lock<std::mutex> lk_dead(mtx_dthreads);
				deadthreads.push_back(th);
				lk_dead.unlock();
			}
			break;
		}
	}
}

void threadpool::stop()
{
	running = false;

	unique_lock<std::mutex> lk_threads(mtx_threads);
	threads.erase(threads.begin(), threads.end());
	lk_threads.unlock();


	unique_lock<std::mutex> lk_dead(mtx_dthreads);
	deadthreads.erase(deadthreads.begin(), deadthreads.end());
	lk_dead.unlock();

}
```

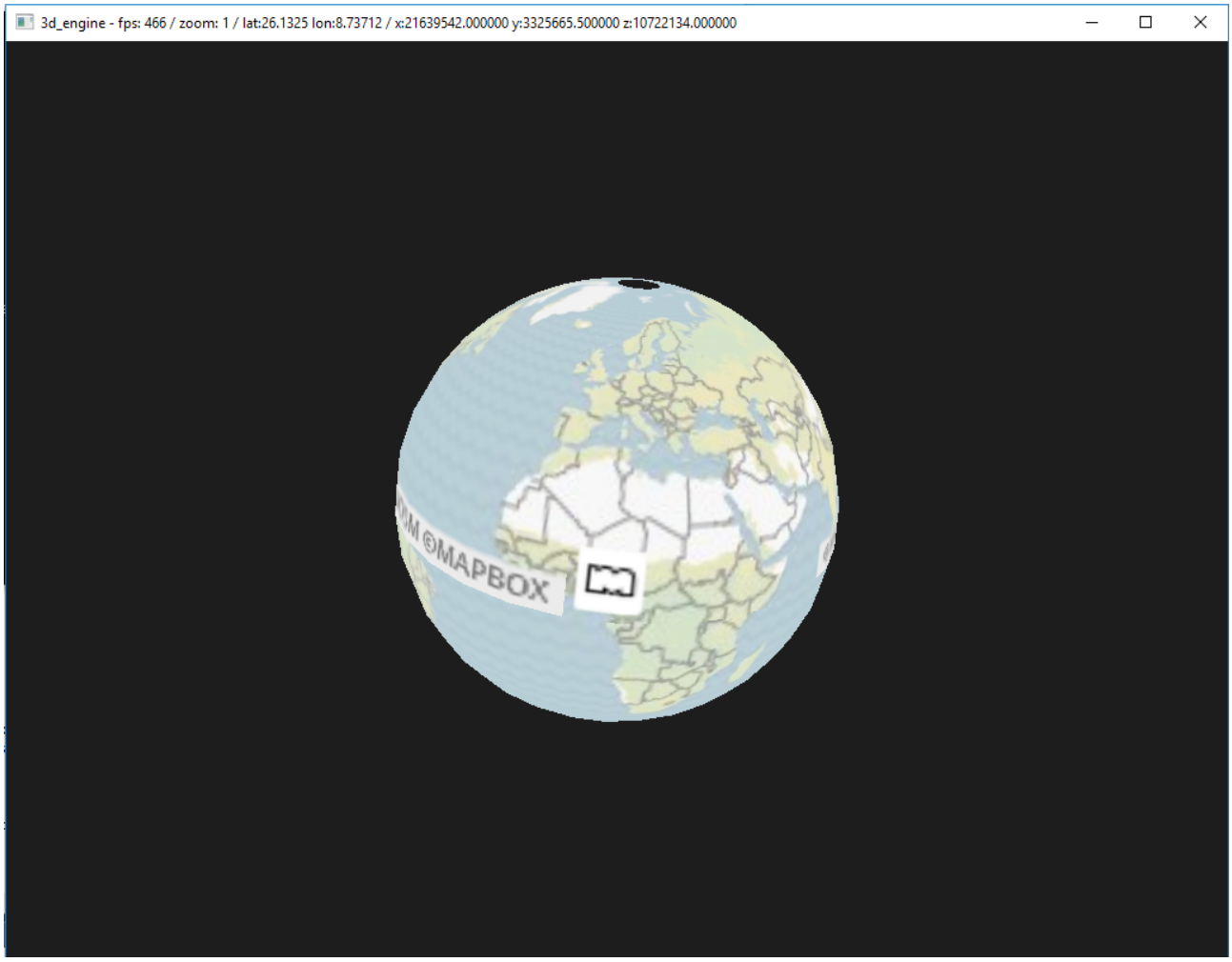
Fakat OpenGL farklı thread ler üzerinden komut kabul etmediğinden hepsi ana thread in ulaşacağı bir liste üzerinde tutulmakta

```
void tilerequest::perform()
{
	shared_ptr<quadtile> tile;
	tile.reset(new quadtile());
	tile->quadkey = quadkey;

	tile->buildplates();

	unique_lock<std::mutex> lk(eng.sc->earth->mxpreparedtiles);
	eng.sc->earth->preparedtiles.push_back(tile);
}
```

ve yükleme işlemleri ana thread üzerinden yapılmaktadır.

```
vector<quadtile*> spheroid::getdisplayedtiles(glm::vec3 cameraPos, int zoomlevel)
{
	vector<quadtile*> dt = tiles.calculatesubtiles(cameraPos, zoomlevel);

	//attach all loaded tiles
	unique_lock<std::mutex> lk(mxpreparedtiles);
	for (auto pt : preparedtiles)
	{
		quadtile* t = tiles.gettile(pt->quadkey);
		if (t == nullptr)
			continue;

		t->normals = pt->normals;
		t->uvs = pt->uvs;
		t->vertices = pt->vertices;
		string fname(pt->fname.begin(), pt->fname.end());
		t->tm.reset(new texturemesh(t->vertices, t->normals, t->uvs, fname));
	}
	preparedtiles.clear();


	return dt;
}
```

Bu noktaya kadar düşünüldüğünde elimizde temel manada aşağıdaki görülen sistem oluşmaktadır.



Bu noktadan sonra yapılacak iş ise zoom sistemi geliştirmek. Kullanıcı her yaklaşma yaptığında tüm dünya haritasının güncellenmesi mümkün değildir çünkü en yüksek yaklaşma seviyesi olan 19. seviyede 4^19 alt karede her biri 256x256 pixel resim gerektirmektedir. Bu noktada yapılması gereken kullanıcının nereye yaklaştığını bulup sadece bu bölgeyi güncellemek.

Bu konu için burada kullanacağımız metod şu şekildedir. Elimizde kameranın matematiksel konumu bulunmaktadır Bu konumundan dünyaya atılacak biş ışının yeryüzünde hangi noktayı kestiğini ve bu noktaların enlem boylamını bulmak pekala mümkündür.

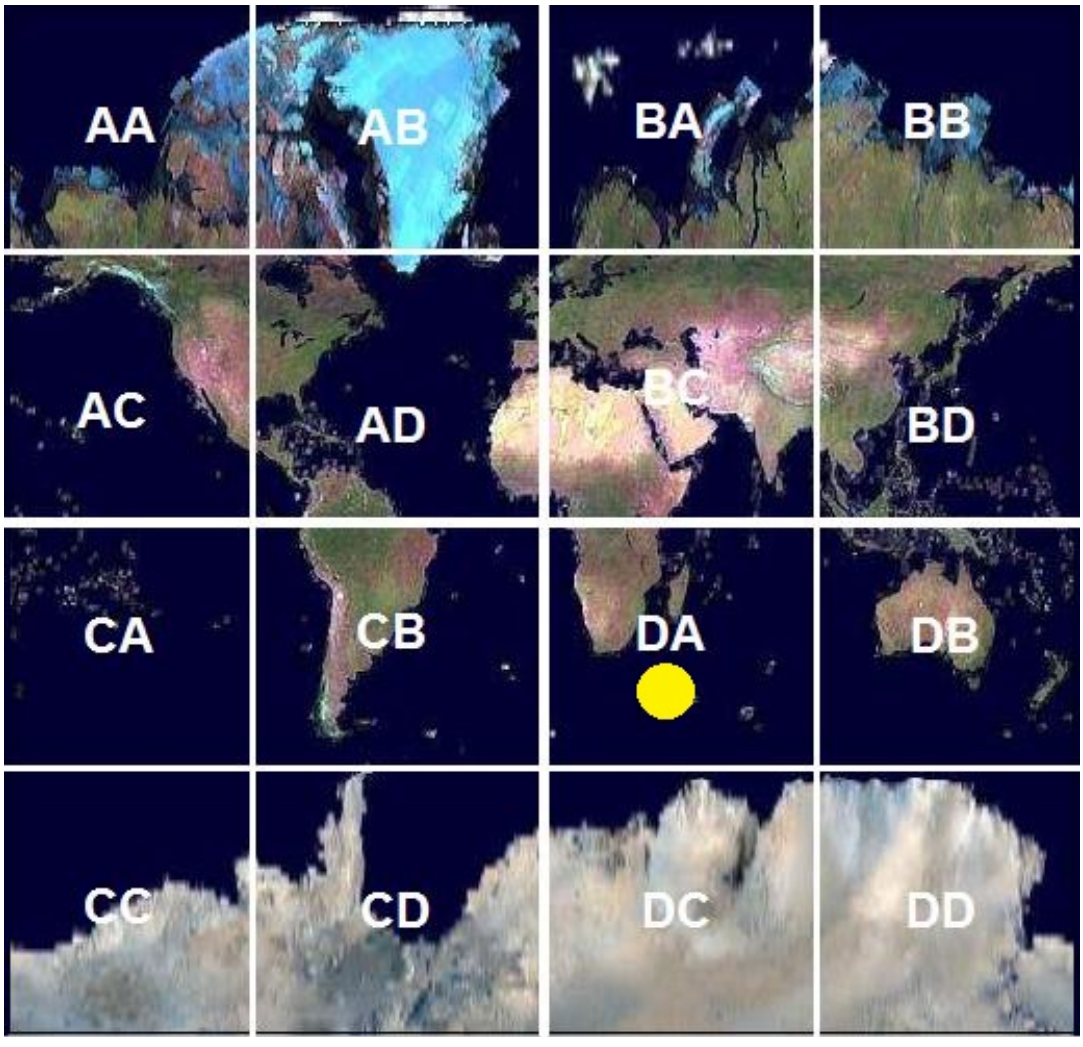
```
std::array<double, 3> ecef_to_geo(std::array<double,3> ecef) {

	double  a = 6378137.0;              //WGS-84 semi-major axis
	double e2 = 6.6943799901377997e-3;  //WGS-84 first eccentricity squared
	double a1 = 4.2697672707157535e+4;  //a1 = a*e2
	double a2 = 1.8230912546075455e+9;  //a2 = a1*a1
	double a3 = 1.4291722289812413e+2;  //a3 = a1*e2/2
	double a4 = 4.5577281365188637e+9;  //a4 = 2.5*a2
	double a5 = 4.2840589930055659e+4;  //a5 = a1+a3
	double a6 = 9.9330562000986220e-1;  //a6 = 1-e2
	double zp, w2, w, r2, r, s2, c2, s, c, ss;
	double g, rg, rf, u, v, m, f, p, x, y, z;
	double n, lat, lon, alt;
```

```
	std::array<double,3> geo;	//Results go here (Lat, Lon, Altitude)
	x = ecef[0];
	y = ecef[1];
	z = ecef[2];
	zp = abs(z);
	w2 = x * x + y * y;
	w = sqrt(w2);
	r2 = w2 + z * z;
	r = sqrt(r2);
	geo[1] = atan2(y, x);		//Lon (final)
	s2 = z * z / r2;
	c2 = w2 / r2;
	u = a2 / r;
	v = a3 - a4 / r;
	if (c2 > 0.3) {
		s = (zp / r)*(1.0 + c2 * (a1 + u + s2 * v) / r);
		geo[0] = asin(s);	//Lat
		ss = s * s;
		c = sqrt(1.0 - ss);
	}
	else {
		c = (w / r)*(1.0 - s2 * (a5 - u - c2 * v) / r);
		geo[0] = acos(c);	//Lat
		ss = 1.0 - c * c;
		s = sqrt(ss);
	}
	g = 1.0 - e2 * ss;
	rg = a / sqrt(g);
	rf = a6 * rg;
	u = w - rg * c;
	v = zp - rf * s;
	f = c * u + s * v;
	m = c * v - s * u;
	p = m / (rf / g + f);
	geo[0] = geo[0] + p;		//Lat
	geo[2] = f + m * p / 2.0;	//Altitude
	if (z < 0.0) {
		geo[0] *= -1.0;		//Lat
	}

	geo[0] = geo[0] * 180 / glm::pi<double>();
	geo[1] = geo[1] * 180 / glm::pi<double>();

	return(geo);	//Return Lat, Lon, Altitude in that order
}
```

Şimdi bulduğumuz bu noktanın 1024x1024 bir mercator dünya haritası üzerinde hangi noktaya geldiğini hesap edelim.

```
double lon2mercx(double lon, double mapsize = 1024)
{
	//lon range is -180 to 180
	return mapsize * ((lon + 180) / 360);
}

double lat2mercy(double lat, double mapsize = 1024)
{
	if (lat < -85.08)
		lat = -85.08;

	float a = 6378137.0f;
```

```
        float b = 6356752.3f;

        float e2 = 1 - ((b / a) * (b / a));
        float e = sqrt(e2);

        double phi = lat * glm::pi<double>()/180;
        double sinphi = sin(phi);
        double con = e * sinphi;

        con = pow((1.0 - con) / (1.0 + con), e/2);

        double ts = tan(0.5 * (glm::pi<double>() * 0.5 - phi)) / con;

        double distance = 0 - a * log(ts);
        double circumference = 2 * glm::pi<double>() * 6378137.0f;
        //0 to circumference
        distance += circumference / 2;
        //now take the complement.
        distance = circumference - distance;

        distance = (distance / circumference) * mapsize;

        //sanity check
        distance = distance < 0 ? 0 : distance;
        distance = distance > mapsize ? mapsize : distance;

        return distance;
}
```



Bir örnek üzerinde bulduğumuz noktanın yukarıda bulunan haritadaki sarı noktaya denk geldiğini hayal edelim.

Bu aşamada elimizdeki bütün karelerin teker teker sarı noktayı içerip içermediğini hesap etmemiz mümkün. Matematiksel olarak bu tarz formüller bulunabilir. Burada dikkat etmemiz gereken husus 19. seviyede 274877906944 adet kare bulunduğu 30 kare/saniye gösterimde bu tam olarak saniyede 30 x 274877906944 karenin kontrol edilmesi anlamına gelir. Bu kadar işlem gücünü temin edebilsek dahi sistemin tüketeceği enerjiyi de dikkate almalıyız çünkü batarya ile çalışan cihazlar için bu ciddi sıkıntılar doğuracaktır.

Bu sayılan problemlerin çözümü için kullanacağımız yöntem ise biraz farklı. Yukarıdaki gibi 2. yaklaşma seviyesi için önceliki olarak noktanın A,B,C,D dörtlüsünün hangisinin içinde kaldığını bulmamız gerekiyor. Bunu basit olarak noktanın A,B,C,D nin kenarlarına olan uzaklığını hesaplayarak bulabiliriz. En kısa olan ağacı kendi içinde dörde bölüp alt düğümler için denediğimizde cevabın D → A olduğunu bulmamız mümkün.

Sadece bakılan karenin bulunması yeterli gelmemektedir. Etrafındaki karelerinde ekranda bulunacağı göz önüne alındığında bu karelerin neler olduğuda hesaplanmalıdır. Bunu hesaplayabilmek için kullanacağımız yöntem ise quadkey girdisini koordinat çevirmek.

```
std::array<int, 2> tile2pos(string quadkey)
{
	array<int, 2> pos = {0,0};

	int x = 0;
	int y = 0;
	for (size_t i = 0; i < quadkey.size(); i++)
	{
		pos[0] += ((quadkey.at(i) - 65) & 1) * pow(2, quadkey.size() - (i+1));
		pos[1] += ((quadkey.at(i) - 65) / 2) * pow(2, quadkey.size() - (i+1));
	}

	return pos;
}

string pos2tile(int x, int y, int zoomlevel)
{
	if (zoomlevel < 2)
		return "";

	string quadkey = "";

	//no solution for these yet
	y = y == -1 ? 0 : y;
	y = y >= pow(2, zoomlevel) ? pow(2, zoomlevel) - 1 : y;



	if (x == pow(2, zoomlevel))
		x = 0;
	else if (x < 0)//-1 situation
		x = pow(2, zoomlevel) - 1;



	for (size_t i = 0; i < zoomlevel; i++)
```

```
	{
		int sc = pow(2, zoomlevel-(i+1));
		int t = 0;

		//A-C or B-D?
		if (x >= sc)
		{
			t += 1;
			x = x % sc;
		}

		//if A-C then A or C, if B-D pair then B or D?
		if (y >= sc)
		{
			t += 2;
			y = y % sc;
		}

		quadkey += (char)(65+t);
	}
	return quadkey;
}
```

ve etrafındaki 1 sıra karenin koordinatlarını oluşturup sonra tekrardan bunu quadkey biçimine dönüştürmek

```
	for (int x = -1; x < 2; x++)
	{
		for (int y = -1; y < 2; y++)
		{
			string tile = pos2tile(c[0] + x, c[1] + y, q.size());
			if (tile.size() > 0)
				surroundingtiles.insert(tile);
		}
	}
```

Bu işlemlerin ardından artık hangi karelerin görüntüye gireceğini tespit ettik fakat, bu yapraklara ulaşan yolu quadtree açmamız gerekir bu iş için aşağıdaki işlevi kullanacağız.

```
void quadtile::germinate(string tile)
{
	if (tile.size() == 0)
		return;

	quadtile* child = getchild(tile.at(0));
	if (child == nullptr)
	{
		initchildren();
		child = getchild(tile.at(0));
	}

	for (int i = 0; i < 4; i++)
	{
		if (!children[i].requested)
```

```
		{
			eng.sc->earth->pool.queue(shared_ptr<tilerequest>(new
tilerequest(children[i].quadkey)));
			children[i].requested = true;
		}
	}

	child->germinate(tile.substr(1));

	return;
}
```

Kullanıcımız dünya üzerinde bir mühlet gezinti yaptığında görüntülediği her yerin detaylı haritalarını sunmak bir süre sonra sistemin aşırı yüklenip yavaşlamasına neden olacağı aşikardır. Bu yüzden görüntülenmeyen kısımlara yüklenmiş detaylı haritaların kaldırılıp daha düşük çözünürlüklü haritalarla yenilenmesi makul bir fikirdir. Dolayısıyla ağacımız filizlenmekle birlikte budanmalıdır da.

```
void quadtile::prune(string root, set<string>& sprouts, int zoomlevel)
{
	if (children == nullptr)
		return;

	for (size_t i = 0; i < 4; i++)
	{
		char cc = 65 + i;
		string leaf = root + cc;
		bool found = false;

		for (auto tile : sprouts )
		{
			if (tile.find(leaf) == 0)
			{
				found = true;
				break;
			}
		}

		if (found)
		{
			quadtile* c = getchild(cc);

			if (c != nullptr && leaf.size() < zoomlevel)
				prune(leaf, sprouts, zoomlevel);
			else
				invalidate(leaf);
		}
		else
		{
			invalidate(leaf);
		}
	}
}
```

Sistemimiz bu haliyle izlendiğinde bir noktaya yaklaşıldığında bu noktanın karardığı görülebilir çünkü biz bir noktayı detaylı göstermek istediğimizde o plakayı yok edip, yerine aynı boyutta 4 alt plaka oluşturup, internet üzerinden talep ettiğimiz haritalarla kaplıyoruz. Bu süreç boyunca bu bölge siyah olarak kalıyor ama ağaç üzerinde dolaşarak bir plakanın alt 4 plakası da tam

olarak hazır değilse üst plakayı göstermeye devam et dersek, pusludan net görüntüye geçiş mümkündür.

```
set<quadtile*> getcompletetree(quadtile* root)
{
	set<quadtile*> tiles;


	if (root->children == nullptr)
	{
		tiles.insert(root);
		return tiles;
	}

	int ltc = 0;
	for (int i = 0; i < 4; i++)
	{
		if (root->children[i].tm != nullptr)
		{
			ltc++;
		}
	}

	if (ltc < 4)
	{
		if(root->quadkey.size() > 0)
			tiles.insert(root);

		return tiles;
	}


	for (int i = 0; i < 4; i++)
	{
		set<quadtile*> subtiles = getcompletetree(&root->children[i]);
		tiles.insert(subtiles.begin(), subtiles.end());
	}

	return tiles;
}
```

3. SONUÇ

Böylelikle sistemimiz kaba hatlarıyla ortaya çıkmış bulunmaktadır. Sistem üzerinde 3 boyutlu olarak yeryüzü şekillerinin gösteriminden dünya etrafındaki uydular ve hatta hava sahaları üzerinde uçuş yapan herhangi bir öğeyi göstermeye kadar hemen her türlü ekleme yapılabilir. Sadece havada değil aynı zamanda yüzey üzerinde ormanlar, yağmur bulutları , yer altı kaynakları gibi 3 boyutlu coğrafi verilerin gösterilebilmesi de yine ek çalışmalar aracılığıyla yapılabilir. Bize ayrılan süre ancak bu kadar olduğundan biz projemizi bu noktada sonlandıracağız.

Projemizin bütün kodları http://github.com/ademirtug/digital_globe/ websitesi üzerinde mevcut bulunmaktadır.

4. KAYNAKÇA

[1] D. Rose - http://danceswithcode.net/engineeringnotes/geodetic_to_ecef/geodetic_to_ecef.html

[2] Marcin Ligas, Piotr Banasik - http://www.iag-aig.org/attach/989c8e501d9c5b5e2736955baf2632f5/V60N2_5FT.pdf

[3] Open Street Maps - https://wiki.openstreetmap.org/wiki/QuadTiles

[4] Mapzen - https://mapzen.com/blog/escape-from-mercator/

[5] Open Street Maps - https://wiki.openstreetmap.org/wiki/Mercator#C_implementation

[6] Wikipedia - https://en.wikipedia.org/wiki/Mercator_projection

[7] Microsoft - https://docs.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system

[8] ABD İçişleri Bakanlığı - https://pubs.usgs.gov/pp/1395/report.pdf