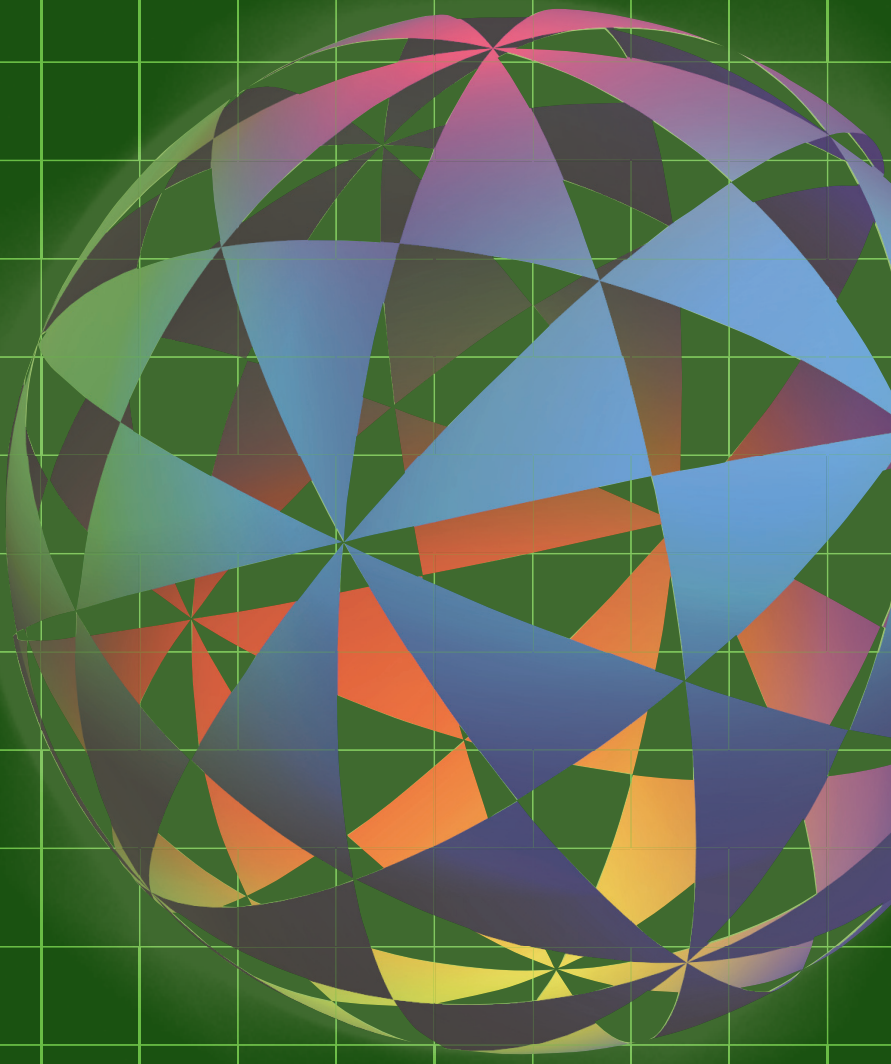
Przemysław Kiciak

OpenGL i GLSL

(nie taki krótki kurs)

Część II



PWN

OpenGL i GLSL

Przemysław Kiciak

OpenGL i GLSL

(nie taki krótki kurs)

Część II

PWN

Projekt okładki i stron tytułowych Anna Ludwicka
Wydawca Wioleta Szczygielska-Dybciak
Redaktor prowadzący Jolanta Kowalczuk
Redaktor Maria Kasperska
Skład systemem TEX Przemysław Kiciak
Koordynator produkcji Anna Bączkowska

Recenzenci
prof. dr hab. Krzysztof Diks
prof. dr hab. inż. Przemysław Rokita

ISBN 978-83-01-20863-9 cz. II
ISBN 978-83-01-20866-0 cz. I–III

Wydanie I
Warszawa 2019

Wydawnictwo Naukowe PWN
02-460 Warszawa, ul. Gottlieba Daimlera 2
tel. (0-22) 69 54 321, faks 22 69 54 288
infolinia 801 33 33 88
e-mail: pwn@pwn.com.pl, reklama@pwn.pl
www.pwn.pl

Druk i oprawa: Partner Poligrafia Andrzej Kardasz

Spis treści części II

14

Druga aplikacja

Druga aplikacja rysuje powierzchnie zakrzywione zbudowane z płatów Béziera. Dla odmiany i poszerzenia horyzontów jest to aplikacja biblioteki GLFW. Wykorzystamy w niej wiele procedur z pierwszej aplikacji, bez żadnych zmian lub ze zmianami wymuszonymi przez inny interfejs aplikacji (API) tej biblioteki. Ale to nie są wielkie zmiany.

14.1. Płaty powierzchni Béziera

O binómio de Newton é tão belo como a Vénus de Milo.
O que há é pouca gente para dar por isso.[1]

Fernando Pessoa

Opis reprezentacji Béziera wielomianowych płatów parametrycznych i jej własności można znaleźć w książkach na ten temat, na przykład w mojej [25]. Matematyka będąca podstawą tej reprezentacji może wystraszyć wiele osób[2], więc nie przedstawiam tu jej zbyt szczegółowo, jednak bez tej matematyki nie ma mowy o eleganckich i efektywnych algorytmach umożliwiających wykonywanie obrazów takich płatów i w będących w moim posiadaniu książkach o OpenGL-u nie takie algorytmy są opisane. Poniższy (zgniły) kompromis wystarczy do przedstawienia algorytmów, których użyjemy, a Czytelników zachęcam do dowiedzenia się więcej z innych źródeł.

Podstawą reprezentacji Béziera krzywych i płatów parametrycznych są **wielomiany bazowe Bernsteina**, określone wzorem

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0, \ldots, n. \tag{14.1}$$

Krzywą Béziera stopnia n określa się wzorem

$$\boldsymbol{p}(t) = \sum_{i=0}^{n} \boldsymbol{p}_i B_i^n(t), \quad t \in [a, b]. \tag{14.2}$$

[1] Dwumian Newtona jest równie piękny jak Wenus z Milo. Jak niewielu ludzi zdaje sobie z tego sprawę.
[2] nad czym głęboko ubolewam

Odcinek (przedział) $[a,b]$ osi liczbowej jest **dziedziną parametryzacji** $\boldsymbol{p}$. Każdej liczbie t z tego przedziału odpowiada pewien punkt $\boldsymbol{p}(t)$ krzywej. Krzywa ta znajduje się w przestrzeni, do której należą tzw. **punkty kontrolne** $\boldsymbol{p}_0,\dots,\boldsymbol{p}_n$. Dla każdego $t\in\mathbb{R}$ jest spełniona równość $\sum_{i=0}^{n} B_i^n(t)=1$, dzięki czemu każdy punkt $\boldsymbol{p}(t)$ jest kombinacją afiniczną punktów kontrolnych krzywej.

Przedział $[a,b]$ może być wybrany dowolnie, ale zazwyczaj przyjmuje się, że jest to przedział $[0,1]$. Wszystkie wielomiany Bernsteina przyjmują w nim wartości nieujemne, co w powiązaniu z faktem, że ich suma dla każdego t jest równa 1, sprawia, że jeśli $[a,b]=[0,1]$, to krzywa jest położona w **otoczce wypukłej** zbioru swoich punktów kontrolnych.

Łamana kontrolna składa się z n odcinków, a jej kolejnymi wierzchołkami są punkty $\boldsymbol{p}_0,\dots,\boldsymbol{p}_n$. Jest ona przybliżeniem krzywej Béziera; ponieważ $\boldsymbol{p}(0)=\boldsymbol{p}_0$ oraz $\boldsymbol{p}(1)=\boldsymbol{p}_n$, punktami końcowymi krzywej są pierwszy i ostatni wierzchołek łamanej kontrolnej. Jeśli f oznacza dowolne przekształcenie afiniczne, to obraz $f(\boldsymbol{p})$ krzywej $\boldsymbol{p}$ jest reprezentowany przez punkty kontrolne $f(\boldsymbol{p}_0),\dots,f(\boldsymbol{p}_n)$. Zatem, aby poddać krzywą Béziera dowolnemu przekształceniu afinicznemu (np. obrócić ją lub przesunąć), wystarczy zastosować to przekształcenie do jej punktów kontrolnych.

Tensorowy płat Béziera stopnia (n,m) jest określony wzorem

$$\boldsymbol{p}(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m}\boldsymbol{p}_{ij}B_i^n(u)B_j^m(v),\qquad u\in[a,b],\ v\in[c,d]. \tag{14.3}$$

Dziedziną parametryzacji płata jest zatem prostokąt $[a,b]\times[c,d]$, przy czym zazwyczaj przyjmuje się, że $a=c=0$, $b=d=1$, a wtedy dziedzina ta jest kwadratem jednostkowym, $[0,1]^2$. Krzywą można widzieć jako powyginany i porozciągany odcinek i podobnie płat tensorowy jest powyginanym i porozciąganym kwadratem.

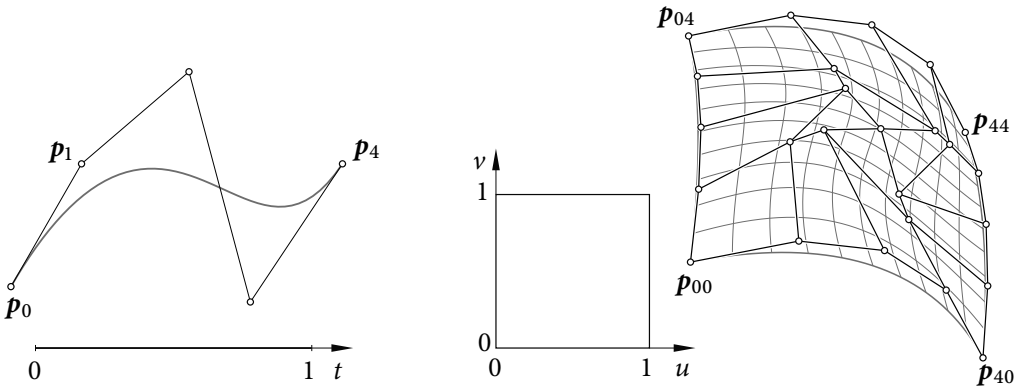
W zbiorze **punktów kontrolnych płata**, $\boldsymbol{p}_{ij}$, wyróżniamy $n+1$ **kolumn** (ciągów $\boldsymbol{p}_{i0},\dots,\boldsymbol{p}_{im}$) oraz $m+1$ **wierszy** (ciągów $\boldsymbol{p}_{0j},\dots,\boldsymbol{p}_{nj}$), które wygodnie jest przedstawiać jako łamane. W ten sposób powstaje **siatka kontrolna** — odpowiednik łamanej kontrolnej krzywej. Jej kształt określa kształt płata. Podobnie jak dla krzywej, aby otrzymać obraz płata Béziera w dowolnym przekształceniu afinicznym, wystarczy przekształcić jego siatkę kontrolną.

Wzór definiujący płat możemy przepisać w postaci

$$\boldsymbol{p}(u,v)=\sum_{i=0}^{n}\underbrace{\left(\sum_{j=0}^{m}\boldsymbol{p}_{ij}B_j^m(v)\right)}_{\boldsymbol{q}_i}B_i^n(u)=\sum_{i=0}^{n}\boldsymbol{q}_iB_i^n(u). \tag{14.4}$$

Wynika z niej, że mając dane liczby u,v, możemy obliczyć $n+1$ punktów, $\boldsymbol{q}_0,\dots,\boldsymbol{q}_n$, z których każdy jest punktem krzywej Béziera stopnia m reprezentowanej przez odpowiednią kolumnę siatki kontrolnej. Otrzymamy w ten sposób **krzywą stałego parametru** v płata; punkt $\boldsymbol{p}(u,v)$ płata jest punktem tej krzywej, odpowiadającym danemu u.[3]

[3] Alternatywnie, zamiast kolumn możemy potraktować *wiersze* siatki kontrolnej jak łamane kontrolne krzywych Béziera stopnia n; otrzymamy w ten sposób reprezentację Béziera krzywej stałego parametru u płata i możemy obliczać punkty płata jako punkty tej krzywej.



Rysunek 14.1. Krzywa Béziera i tensorowy płat Béziera

Zanim zajmiemy się algorytmami znajdowania punktów krzywych Béziera, zbadajmy wynikające z powyższych spostrzeżeń (niektóre) własności płatów i (niektóre) ich konsekwencje praktyczne. Zauważamy, że

$$\boldsymbol{p}(u,0) = \sum_{i=0}^{n} \boldsymbol{p}_{i0} B_i^n(u), \quad \boldsymbol{p}(u,1) = \sum_{i=0}^{n} \boldsymbol{p}_{im} B_i^n(u),$$
$$\boldsymbol{p}(0,v) = \sum_{j=0}^{m} \boldsymbol{p}_{0j} B_j^m(v), \quad \boldsymbol{p}(1,v) = \sum_{j=0}^{m} \boldsymbol{p}_{nj} B_j^m(v),$$

a zatem skrajne wiersze i kolumny siatki kontrolnej wyznaczają cztery krzywe brzegowe płata o dziedzinie $[0,1]^2$, co więcej, narożniki siatki kontrolnej są narożnikami płata: $\boldsymbol{p}(0,0) = \boldsymbol{p}_{00}$, $\boldsymbol{p}(1,0) = \boldsymbol{p}_{n0}$, $\boldsymbol{p}(0,1) = \boldsymbol{p}_{0m}$, $\boldsymbol{p}(1,1) = \boldsymbol{p}_{nm}$.

Do wykonania obrazu potrzebny jest algorytm obliczania punktu $\boldsymbol{p}(u,v)$ dla danych liczb u, v oraz wektora $\boldsymbol{n}(u,v) = \boldsymbol{p}_u(u,v) \wedge \boldsymbol{p}_v(u,v)$, tj. iloczynu wektorowego pochodnych cząstkowych parametryzacji $\boldsymbol{p}$, który jest wektorem normalnym płata w punkcie $\boldsymbol{p}(u,v)$; wektor ten natychmiast po obliczeniu unormujemy, aby otrzymać wektor jednostkowy. Potrzebujemy zatem algorytmu obliczania pochodnych cząstkowych płata Béziera. Wyprowadzanie go zaczniemy od spojrzenia na krzywe.

Aby obliczać punkty krzywej Béziera, oznaczmy $s = 1 - t$ i rozpiszmy wzór (14.2)[4]:

$$\begin{aligned}\boldsymbol{p}(t) &= \boldsymbol{p}_0\tbinom{n}{0}s^n + \boldsymbol{p}_1\tbinom{n}{1}ts^{n-1} + \cdots + \boldsymbol{p}_{n-1}\tbinom{n}{n-1}t^{n-1}s + \boldsymbol{p}\tbinom{n}{n}t^n \\ &= \Big(\ldots\big(\boldsymbol{p}_0\tbinom{n}{0}s + \boldsymbol{p}_1\tbinom{n}{1}t\big)s + \cdots + \boldsymbol{p}_{n-1}\tbinom{n}{n-1}t^{n-1}\Big)s + \boldsymbol{p}\tbinom{n}{n}t^n.\end{aligned}$$

Na podstawie tego wzoru możemy obliczyć punkt $\boldsymbol{p}(t)$ jako wartość (wektorowego) wielomianu zmiennej s, korzystając ze schematu Hornera. W tym celu trzeba obliczyć (wektorowe) współczynniki $\boldsymbol{p}_i\binom{n}{i}t^i$ tego wielomianu w bazie potęgowej. Punkty kontrolne $\boldsymbol{p}_i$

[4]Na początku otwieramy $n-1$ nawiasów, z których każdy zamkniemy po dodaniu kolejnego składnika o postaci $\boldsymbol{p}_i\binom{n}{i}t^i$, a po zamknięciu wyrażenie w nawiasie pomnożymy przez s.

mamy dane, kolejne potęgi parametru t obliczymy w pętli, a do obliczenia współczynników dwumianowych Newtona możemy użyć wzorów

$$\binom{n}{0} = 1, \quad \binom{n}{1} = n, \quad \binom{n}{i+1} = \binom{n}{i}(n-i)/(i+1), \ i = 1, \ldots, n-1.$$

Współczyniki dwumianowe są liczbami całkowitymi; w szczególności iloczyn $\binom{n}{i}(n-i)$ jest zawsze podzielny przez $i+1$ i tu w działaniach stałopozycyjnych nie ma błędów zaokrągleń. Ale może być nadmiar — dla 32-bitowych liczb całkowitych ze znakiem wystąpi on, gdy $n \geqslant 30$. W praktycznych zastosowaniach mamy do czynienia z krzywymi i płatami znacznie niższych stopni.

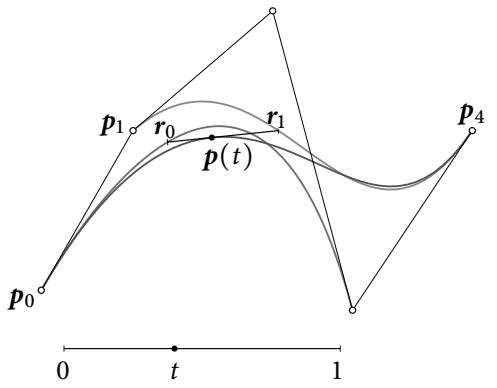
Aby znaleźć punkt krzywej i wektor pochodnej parametryzacji krzywej Béziera, możemy skorzystać z faktu, że parametryzacja określona wzorem (14.2) dla $n > 0$ ma równoważne przedstawienie w postaci

$$\boldsymbol{p}(t) = (1-t)\underbrace{\sum_{i=0}^{n-1} \boldsymbol{p}_i B_i^{n-1}(t)}_{\boldsymbol{r}_0} + t\underbrace{\sum_{i=0}^{n-1} \boldsymbol{p}_{i+1} B_i^{n-1}(t)}_{\boldsymbol{r}_1} = (1-t)\boldsymbol{r}_0 + t\boldsymbol{r}_1, \tag{14.5}$$

a pochodna parametryzacji w punkcie t wyraża się wzorem

$$\boldsymbol{p}'(t) = \sum_{i=0}^{n-1} n(\boldsymbol{p}_{i+1} - \boldsymbol{p}_i) B_i^{n-1}(t) = n(\boldsymbol{r}_1 - \boldsymbol{r}_0). \tag{14.6}$$

Obliczenie punktu $\boldsymbol{p}(t)$ i wektora $\boldsymbol{p}'(t)$ dla danego t można zatem zacząć od znalezienia punktów $\boldsymbol{r}_0$ i $\boldsymbol{r}_1$ położonych na dwóch krzywych Béziera stopnia $n-1$: pierwsza z nich jest reprezentowana przez punkty kontrolne $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_{n-1}$, a druga przez punkty $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$ (rys. 14.2).



Rysunek 14.2. Obliczanie punktu i wektora pochodnej krzywej Béziera

Teraz zobaczmy, jak można obliczyć punkt tensorowego płata Béziera i pochodne cząstkowe jego parametryzacji. Pochodna cząstkowa względem u jest pochodną krzywej stałego parametru v, będącej krzywą Béziera reprezentowaną przez punkty $\boldsymbol{q}_0, \ldots, \boldsymbol{q}_n$ (zobacz

wzór (14.4)). Natomiast pochodną cząstkową względem v możemy otrzymać, jeśli oprócz punktów $\boldsymbol{q}_i$ (będących punktami krzywych Béziera stopnia m reprezentowanych przez kolumny siatki kontrolnej płata) dla ustalonego parametru v obliczymy wektory $\boldsymbol{q}'_0, \ldots, \boldsymbol{q}'_n$ — pochodne tych krzywych w punkcie v. Mamy zatem

$$\boldsymbol{p}_u(u,v) = n\sum_{i=0}^{n-1}\sum_{j=0}^{m}(\boldsymbol{p}_{i+1,j} - \boldsymbol{p}_{ij})B_i^{n-1}(u)B_j^m(v) = n(\boldsymbol{r}_1 - \boldsymbol{r}_0), \tag{14.7}$$

$$\boldsymbol{p}_v(u,v) = m\sum_{i=0}^{n}\sum_{j=0}^{m-1}(\boldsymbol{p}_{i,j+1} - \boldsymbol{p}_{ij})B_i^n(u)B_j^{m-1}(v) = \sum_{i=0}^{n}\boldsymbol{q}'_i B_i^n(u), \tag{14.8}$$

przy czym punkty $\boldsymbol{r}_0$ i $\boldsymbol{r}_1$ znajdziemy podczas obliczania punktu $\boldsymbol{p}(u,v)$ na krzywej stałego parametru v.

Obliczenia pochodnych cząstkowych płata Béziera możemy nie dokończyć, pomijając mnożenie przez czynniki n i m; pochodne są potrzebne do obliczenia jednostkowego wektora normalnego płata, a zatem ich długości są nieistotne — istotne są tylko kierunki i zwroty tych wektorów, bo natychmiast po obliczeniu ich iloczynu wektorowego (przy użyciu dostępnej w GLSL-u funkcji `cross`) iloczyn ten unormujemy. Implementacje w GLSL-u algorytmów tu opisanych są pokazane na listingach 14.5 i 14.6.

14.2. Wymierne płaty Béziera

Wspomnijmy w tym miejscu o **wymiernych płatach Béziera**. Otrzymujemy je, konstruując płaty wielomianowe (takie jak opisane wyżej) w przestrzeni $\mathbb{R}^4$; ich punkty kontrolne są wektorami współrzędnych jednorodnych punktów w $\mathbb{R}^3$. Punkt $\boldsymbol{P}(u,v)$ płata wielomianowego w $\mathbb{R}^4$ (tzw. **płata jednorodnego**) traktujemy jak wektor współrzędnych jednorodnych punktu $\boldsymbol{p}(u,v)$ płata wymiernego w przestrzeni trójwymiarowej. Współrzędne kartezjańskie tego punktu jak zwykle otrzymamy przez podzielenie pierwszych trzech współrzędnych jednorodnych przez czwartą współrzędną (wagową). Jeśli wszystkie punkty kontrolne mają taką samą współrzędną wagową (która nie może być zerem), to mamy zwykły wielomianowy tensorowy płat Béziera. Ale jeśli dopuszczamy niejednakowe współrzędne wagowe punktów kontrolnych, to mamy istotnie szerszą klasę płatów powierzchni[5].

Siatkę kontrolną płata wymiernego możemy narysować jako obiekt trójwymiarowy; w tym celu obliczamy współrzędne jej wierzchołków, dzieląc pierwsze trzy współrzędne przez współrzędną wagową. Modelując płaty wymierne, postępujemy odwrotnie: rozmieszczamy punkty kontrolne w przestrzeni trójwymiarowej i dobieramy ich wagi; na podstawie punktu w $\mathbb{R}^3$ i wagi możemy łatwo obliczyć punkty kontrolne płata jednorodnego w $\mathbb{R}^4$.

Dla płata wymiernego również potrzebujemy obliczać punkty i wektory normalne. Podam przepis (bez dowodu), jak to czynić. Jeśli symbole $\boldsymbol{P}(u,v)$, $\boldsymbol{P}_u(u,v)$ i $\boldsymbol{P}_v(u,v)$ oznaczają

[5] Zawierającą m.in. wszystkie kwadryki, tj. powierzchnie drugiego stopnia, a także powierzchnie obrotowe, których tworzące mają parametryzacje wymierne.

odpowiednio punkt płata jednorodnego i jego pochodne cząstkowe w punkcie (u, v), to iloczyn wektorowy *trzech* wektorów w $\mathbb{R}^4$

$$\boldsymbol{N}(u,v) = \boldsymbol{P}(u,v) \wedge \boldsymbol{P}_u(u,v) \wedge \boldsymbol{P}_v(u,v)$$

reprezentuje (jednoznacznie) płaszczyznę styczną do płata wymiernego w $\mathbb{R}^3$. W szczególności, biorąc pierwsze trzy współrzędne tego iloczynu, otrzymamy wektor normalny $\boldsymbol{n}(u,v)$ tej płaszczyzny[6].

Ponieważ biblioteka GLSL-a nie zawiera standardowej funkcji obliczającej iloczyn wektorowy w $\mathbb{R}^4$, trzeba taką funkcję napisać samemu. Dla wektorów $\boldsymbol{a} = (x_{\boldsymbol{a}}, y_{\boldsymbol{a}}, z_{\boldsymbol{a}}, w_{\boldsymbol{a}})$, $\boldsymbol{b} = (x_{\boldsymbol{b}}, y_{\boldsymbol{b}}, z_{\boldsymbol{b}}, w_{\boldsymbol{b}})$, $\boldsymbol{c} = (x_{\boldsymbol{c}}, y_{\boldsymbol{c}}, z_{\boldsymbol{c}}, w_{\boldsymbol{c}})$ jest taki wzór:

$$\boldsymbol{a} \wedge \boldsymbol{b} \wedge \boldsymbol{c} = \begin{bmatrix} -d_{34}y_{\boldsymbol{c}} + d_{24}z_{\boldsymbol{c}} - d_{23}w_{\boldsymbol{c}} \\ d_{34}x_{\boldsymbol{c}} - d_{14}z_{\boldsymbol{c}} + d_{13}w_{\boldsymbol{c}} \\ -d_{24}x_{\boldsymbol{c}} + d_{14}y_{\boldsymbol{c}} - d_{12}w_{\boldsymbol{c}} \\ d_{23}x_{\boldsymbol{c}} - d_{13}y_{\boldsymbol{c}} + d_{12}z_{\boldsymbol{c}} \end{bmatrix},$$

a w nim występują liczby

$$d_{12} = x_{\boldsymbol{a}}y_{\boldsymbol{b}} - y_{\boldsymbol{a}}x_{\boldsymbol{b}}, \quad d_{13} = x_{\boldsymbol{a}}z_{\boldsymbol{b}} - z_{\boldsymbol{a}}x_{\boldsymbol{b}}, \quad d_{14} = x_{\boldsymbol{a}}w_{\boldsymbol{b}} - w_{\boldsymbol{a}}x_{\boldsymbol{b}},$$
$$d_{23} = y_{\boldsymbol{a}}z_{\boldsymbol{b}} - z_{\boldsymbol{a}}y_{\boldsymbol{b}}, \quad d_{24} = y_{\boldsymbol{a}}w_{\boldsymbol{b}} - w_{\boldsymbol{a}}y_{\boldsymbol{b}}, \quad d_{34} = z_{\boldsymbol{a}}w_{\boldsymbol{b}} - w_{\boldsymbol{a}}z_{\boldsymbol{b}}.$$

Iloczyn wektorowy z uwagi na często stosowany symbol „×” jest nazywany po angielsku *cross product*, dlatego wbudowana procedura obliczająca iloczyn wektorowy w $\mathbb{R}^3$ ma nazwę `cross`. Możemy użyć tej samej nazwy dla podprogramu obliczającego iloczyn wektorowy w $\mathbb{R}^4$, ponieważ podprogram ten ma więcej parametrów, a ich typ i typ wyniku różni się od typu parametrów i wyniku funkcji wbudowanej, co umożliwia przeciążenie nazwy. Podprogram wykonujący obliczenie iloczynu wektorowego na podstawie powyższych wzorów jest pokazany na listingu 14.1.

Listing 14.1. Procedura obliczania iloczynu wektorowego w $\mathbb{R}^4$

GLSL

```
vec4 cross ( vec4 a, vec4 b, vec4 c )
{
  float d12, d13, d14, d23, d24, d34;

  d12 = a.x*b.y-a.y*b.x;  d13 = a.x*b.z-a.z*b.x;  d14 = a.x*b.w-a.w*b.x;
  d23 = a.y*b.z-a.z*b.y;  d24 = a.y*b.w-a.w*b.y;  d34 = a.z*b.w-a.w*b.z;
  return vec4 ( -d34*c.y+d24*c.z-d23*c.w, d34*c.x-d14*c.z+d13*c.w,
                -d24*c.x+d14*c.y-d12*c.w, d23*c.x-d13*c.y+d12*c.z );
} /*cross*/
```

[6] Jeśli $\boldsymbol{N}(u,v) = \left[\begin{smallmatrix} \boldsymbol{n} \\ w \end{smallmatrix}\right]$, to punkt $\boldsymbol{q} = \boldsymbol{o} - \frac{w}{\boldsymbol{n}^T\boldsymbol{n}}\boldsymbol{n}$ jest punktem tej płaszczyzny położonym najbliżej początku $\boldsymbol{o}$ układu współrzędnych w $\mathbb{R}^3$ — oczywiście nie musi on leżeć na płacie $\boldsymbol{p}$.

14.3. Szadery

Program szaderów do rysowania płatów Béziera korzysta z mechanizmu rozdrabniania; w pamięci GPU umieścimy tablicę z punktami kontrolnymi i będziemy rysować płat czworokątny. Jego dziedziną jest kwadrat jednostkowy $[0,1]^2$, dokładnie taki, jaki standardowo przyjmuje się za dziedzinę tensorowych płatów Béziera. Etap rozdrabniania podzieli tę dziedzinę na trójkąty, które przekaże do dalszych etapów potoku przetwarzania grafiki. Zadaniem szadera rozdrabniania jest obliczenie, dla podanego wierzchołka trójkątów w rozdrobnionej dziedzinie, odpowiedniego punktu płata Béziera i wektora normalnego płata w tym punkcie.

Opisane w znanych mi książkach o OpenGL-u procedury rysowania płatów Béziera zakładają, że punkty kontrolne płatów są dane w VAO, tj. w tablicy wierzchołków, z której poszczególne wierzchołki z odpowiednimi atrybutami są wybierane przez etap pobierania wierzchołków. Ten mechanizm jest wygodny i elastyczny, ale ma podstawowe ograniczenie: małą maksymalną liczbę wierzchołków płata. Limit liczby wierzchołków płata można poznać, wykonując instrukcję

```
glGetIntegerv ( GL_MAX_PATCH_VERTICES, &n );
```

W implementacji, której używam, płat może mieć co najwyżej 32 wierzchołki, co wystarczy do reprezentowania płatów Béziera stopnia $(5,4)$, ale już nie $(5,5)$.[7] Dlatego zastosujemy inne rozwiązanie: tablicę punktów kontrolnych umieścimy w bloku zmiennych jednolitych, a w VAO umieścimy *dowolne* punkty. Szader rozdrabniania będzie brał punkty kontrolne z tej zmiennej jednolitej, a nie ze zmiennej wbudowanej (tablicy) `gl_in`, której zawartość zignoruje. Tracimy przy tym nieco elastyczności, bo pobieranie wierzchołków musimy oprogramować samemu i raczej nie zrobimy tego dla *wszystkich* reprezentacji liczb (dopuścimy tylko liczby zmiennopozycyjne pojedynczej precyzji — chyba, że ktoś sobie doprogramuje inne możliwości). Z drugiej strony etap pobierania wierzchołków narzuca używanie współrzędnych jednorodnych (wektorów w $\mathbb{R}^4$), a tak możemy osobno oprogramować przetwarzanie płatów w $\mathbb{R}^2$, $\mathbb{R}^3$ i $\mathbb{R}^4$, otrzymując znacznie prostsze i trochę szybsze procedury dla pierwszych dwóch przypadków.

Napiszemy program składający się z pięciu szaderów; wypełnią one wszystkie programowalne etapy w potoku przetwarzania grafiki. Program ten ma umożliwiać rysowanie wielu płatów jednocześnie, przy czym mogą to być wielomianowe płaty płaskie oraz wielomianowe i wymierne płaty w przestrzeni trójwymiarowej. Punkty kontrolne umieścimy w tablicy w bloku zmiennych jednolitych; ponadto umożliwimy użycie dodatkowej tablicy indeksów do tablicy punktów kontrolnych. Dzięki niej można zmniejszyć długość tablicy punktów kontrolnych — przez wyeliminowanie kopii punktów wspólnych dla dwóch lub większej liczby płatów (np. wchodzących w skład wspólnego wiersza lub kolumny siatek kontrolnych płatów mających wspólną krzywą brzegową).

Szader wierzchołków tego programu jest pokazany na listingu 14.2. Wypełnia on obowiązek przypisania czegokolwiek zmiennej wbudowanej `gl_Position`, przy czym dalej jej

[7] Płat stopnia (n,m) ma $(n+1)(m+1)$ punktów kontrolnych.

wartość zostanie zignorowana. Ważniejsze jest przekazanie (w linii 9) **numeru instancji**. Aby spowodować rysowanie, aplikacja wywoła procedurę `glDrawArraysInstanced`, której ostatni parametr określa *liczbę instancji* rysowanego prymitywu: jeśli jest ona równa n, to poszczególne instancje są ponumerowane od 0 do $n-1$. Numer instancji jest podawany szaderowi wierzchołków w zmiennej wbudowanej `gl_InstanceID` (typu `int`), a ten kopiuje ją do zmiennej wyjściowej `VertInstance.instance`.

Listing 14.2. Szader wierzchołków

GLSL

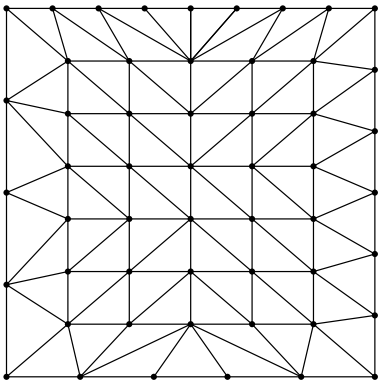
```
1: #version 420 core
2:
3: layout(location=0) in vec4 in_Position;
4:
5: out VertInstance { int instance; } Out;
6:
7: void main ( void )
8: {
9:   Out.instance = gl_InstanceID;
10:  gl_Position = in_Position;
11: } /*main*/
```

Na listingu 14.3 jest pokazany szader sterowania rozdrabnianiem. Jego zadaniem jest przypisanie wartości wszystkim czterem elementom tablicy `gl_TessLevelOuter` i obu elementom tablicy `gl_TessLevelInner`. Wszystkie sześć zmiennych otrzyma tę samą wartość, przy czym zostanie ona wzięta ze zmiennej jednolitej `BezTessLevel`, do której przypisanie wykonała aplikacja[8]. Zmienna ta *nie jest* częścią żadnego nazwanego bloku, a zatem znajduje się w **domyślnym bloku zmiennych jednolitych** programu (p. 9.4). Sposób nadawania wartości takim zmiennym jest przedstawiony dalej, w opisie procedur w C działających na CPU. Tymczasem przypomnijmy, że takie zmienne są widoczne tylko dla *jednego* programu szaderów (choć mają do nich dostęp wszystkie szadery wchodzące w skład tego programu, w których występuje deklaracja takiej zmiennej).

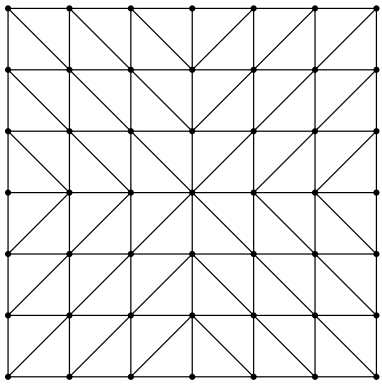
Szader sterowania rozdrabnianiem przekazuje dalej numer instancji i pracowicie kopiuje niepotrzebny dalej wektor współrzędnych jednorodnych wierzchołka. Szader ten ma dostęp do całej tablicy wierzchołków płata (która tu ma długość 4 — linia 3) i jest dla płata wywołany czterokrotnie. Zmienna wbudowana `gl_InvocationID` za każdym razem ma inną wartość, od 0 do 3. Zmienna wyjściowa `Out` musi być tablicą (o długości 4), ale numer instancji wystarczy przekazać tylko w jednym (pierwszym) jej elemencie.

Przykłady podziału dziedziny na trójkąty po podaniu różnych parametrów są pokazane na rysunku 14.3. W ogólności etap rozdrabniania dziedziny nie wytwarza triangulacji Delaunaya (co byłoby świetne, ale chyba zbyt kosztowne, zobacz [31]), ale to, co wytwarza, może być.

[8] Zamiast szadera sterowania rozdrabnianiem możemy użyć procedury `glPatchParameterfv`, zgodnie z opisem w poprzednim rozdziale. Usunięcie szadera sterowania rozdrabnianiem wymagałoby przerobienia szadera wierzchołków lub rozdrabniania w celu uzgodnienia zewnętrznej nazwy zmiennej użytej do przekazania numeru instancji.



gl_TessLevelOuter == {4,5,6,8}
gl_TessLevelInner == {6,7}



gl_TessLevelOuter == {6,6,6,6}
gl_TessLevelInner == {6,6}

Rysunek 14.3. Podziały kwadratowej dziedziny płata

Listing 14.3. Szader sterowania rozdrabnianiem

GLSL

```
#version 420 core

layout (vertices=4) out;

in VertInstance { int instance; } In[];

out TCInstance { int instance; } Out[];

uniform int BezTessLevel;

void main ( void )
{
  if ( gl_InvocationID == 0 ) {
    gl_TessLevelOuter[0] = gl_TessLevelOuter[1] =
    gl_TessLevelOuter[2] = gl_TessLevelOuter[3] = BezTessLevel;
    gl_TessLevelInner[0] = gl_TessLevelInner[1] = BezTessLevel;
    Out[gl_InvocationID].instance = In[gl_InvocationID].instance;
  }
  gl_out[gl_InvocationID].gl_Position = gl_in[gl_InvocationID].gl_Position;
} /*main*/
```

W tej aplikacji nie będziemy zajmować się adaptacyjnym dostosowaniem parametrów rozdrabniania do wielkości obrazu płata, ale warto w tym miejscu uczynić pewną dygresję na ten temat. Zazwyczaj mamy do czynienia z obiektami składającymi się z wielu płatów, czyli z tzw. powierzchniami sklejanymi. Wchodzące w skład takiej powierzchni płaty mają wspólne brzegi, nieraz też są połączone gładko. Podczas rysowania płaty przybliżamy po-

wierzchniami zbudowanymi z (dostatecznie małych) płaskich trójkątów i wtedy brzegi tych płatów są przybliżane łamanymi. Jeśli dwa płaty wchodzące w skład większej powierzchni mają wspólną krzywą brzegową, to łamane przybliżające tę krzywą, złożone z boków odpowiednich trójkątów, muszą być identyczne. W przeciwnym razie powierzchnia przedstawiona na obrazie będzie miała widoczną szczelinę. Pisząc szadery sterowania rozdrabnianiem, które dobierają adaptacyjnie parametry rozdrabniania poszczególnych płatów, trzeba to mieć na uwadze.

Listing 14.4. Szader rozdrabniania — używane zmienne

GLSL

```
#version 420 core

#define MAX_DEG 10

layout(quads,equal_spacing,cw) in;

in TCInstance { int instance; } In[];

out GVertex {
    vec4 Colour;
    vec3 Position;
    vec3 Normal;
  } Out;

uniform bool BezNormals;

uniform CPoints {
    float cp[1];
  } cp;

uniform CPIndices {
    int cpi[1];
  } cpi;

uniform BezPatch {
    int  dim, udeg, vdeg;
    int  stride_u, stride_v, stride_p, stride_q, nq;
    bool use_ind;
    vec4 Colour;
  } bezp;

uniform TransBlock {
    mat4 mm, mmti, vm, pm, mvpm;
    vec4 eyepos;
  } trb;
```

Szader rozdrabniania jest taki długi, że zmieścił się dopiero na pięciu listingach, 14.4–14.8. Pierwszy z nich pokazuje deklaracje wejścia/wyjścia i zmiennych jednolitych używanych przez szader. W linii 5 jest podana informacja (dla etapu rozdrabniania dziedziny), że dziedzina, którą należy podzielić (na trójkąty) jest czworokątem, przy czym boki dziedziny mają być podzielone równomiernie, a orientacja trójkątów przekazywanych dalej ma być zgodna z ruchem wskazówek zegara (`cw` — *clockwise*). Tablica `TCInstance` (linia 7) zawiera w *pierwszym* elemencie numer instancji, który jest interpretowany jako numer płata do narysowania.

Blok wyjściowy `GVertex`, który będzie wejściem dla szadera geometrii, zawiera trzy znajome atrybuty wierzchołka: kolor (który tu pobierzemy z bloku zmiennych jednolitych `BezPatch`) oraz położenie wierzchołka w przestrzeni i wektor normalny płata, podane w układzie współrzędnych świata.

Zmienna jednolita `BezNormals` określa sposób obliczania wektorów normalnych przez program. Jeśli ma ona wartość `true`, to wektor normalny powierzchni ma być obliczany przez szader rozdrabniania i jest to „prawdziwy" wektor normalny powierzchni w obliczonym przez szader punkcie płata. Jeśli wartością tej zmiennej jest `false`, to zadanie to wykona szader geometrii; obliczy on wektor normalny płaszczyzny trójkąta, którego wierzchołki, położone na płacie, obliczył szader rozdrabniania.

W bloku `CPoints` (linie 17–19) jest tablica `cp` ze współrzędnymi punktów kontrolnych. Zadeklarowana długość tablicy jest nieistotna; za jej ustalenie i dopilnowanie, aby program nie wychodził poza zakres jej indeksów, jest odpowiedzialna aplikacja.

Tablica `cpi` w bloku `CPIndices` (linie 21–23) zawiera indeksy do tablicy punktów kontrolnych. Jej faktyczna długość też jest ustalana przez aplikację.

Blok `BezPatch` zawiera opis zbioru płatów Béziera do narysowania. Wartość zmiennej `dim`, 2, 3 lub 4, jest liczbą współrzędnych punktów kontrolnych. Zmienne `udeg` i `vdeg` przechowują stopnie wszystkich płatów w tym zbiorze ze względu na parametry u i v. Zmienne `stride_u`, `stride_v`, `stride_p`, `stride_q` i `nq` służą do uzyskania dostępu do właściwych punktów kontrolnych, co będzie opisane dalej. Zmienna `use_ind` określa, czy po punkty kontrolne (do tablicy `CPoints.cp`) należy sięgać bezpośrednio, czy też posługując się indeksami z tablicy `CPIndices.cpi`. W zmiennej `Colour` jest podany kolor płatów.

Blok `TransBlock` w liniach 32–35, dobrze znany z pierwszej aplikacji, zawiera macierze przekształceń potrzebnych do rzutowania punktu i położenie obserwatora w układzie świata.

Na listingu 14.5 są podane dwie procedury realizujące algorytm obliczania punktu i wektora normalnego płata płaskiego, zawartego w płaszczyźnie xy. Procedura `BCHorner2f` oblicza punkt położony na krzywej stopnia n. Jej punkty kontrolne są podane w tablicy `bcp`. Parametr `t` podaje wartość zadanego argumentu parametryzacji t. Obliczony punkt krzywej ma być przypisany do parametru wyjściowego `p`.

Algorytm zrealizowany w procedurze `BCHorner2f` jest schematem Hornera przedstawionym wcześniej; wartości kolejno przypisywane zmiennej `b` to współczynniki dwumianowe, a zmiennej `d` są przypisywane kolejne potęgi parametru t.

Procedura `BPHorner2f` (linie 16–48) otrzymuje jako parametry `u`, `v` współrzędne u, v punktu w dziedzinie płata. Parametry wyjściowe `pos` i `nv` są zmiennymi, do których należy przypisać obliczony punkt i wektor normalny płata. Zauważmy, że w przypadku płata