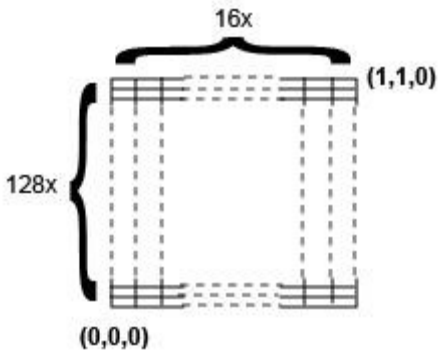
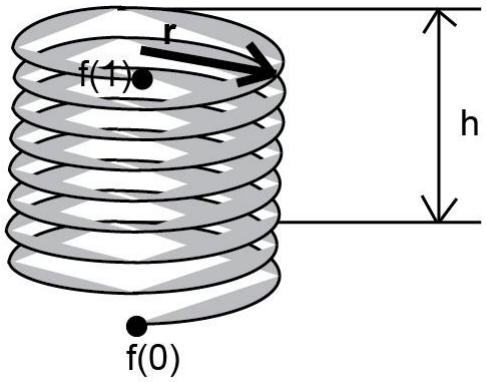
Część III

Generowanie i animacja sprężyny

Wykorzystamy siatkę kwadratu o przeciwległych wierzchołkach (0,0,0) i (1,1,0) podzielonego na 16 kolumn i 128 wierszy (łącznie 4096 trójkątów składających się na kwadrat) jako bazowy materiał, który w shaderze wierzchołków zostanie przekształcony w sprężynę.

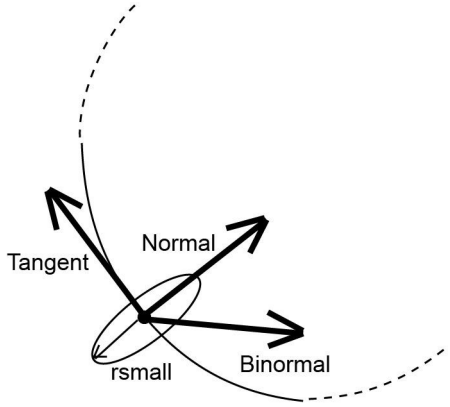


Wykorzystamy parametryczny opis spiralnej krzywej (funkcja przekształcająca przedział [0,1] w oś sprężyny), gdzie współrzędna y wierzchołka siatki posłuży nam za parametr krzywej, aby uzyskać punkt środka przekroju sprężyny, wektor styczny, normalny i binormalny w tym punkcie. Kształt krzywej będzie zależeć od następujących stałych: długości całej krzywej ***l***, wysokości spirali ***h***, promienia spirali ***r***.



Odpowiednia krzywa $f:[0;1] \to R^3$ ma postać $f(s) = \begin{vmatrix} r*\cos(s*a) \\ -h_0 + s*h \\ r*\sin(s*a) \end{vmatrix}$, gdzie h_0 to pewna wartość przesunięcia w pionie, która służy tylko odpowiedniemu umieszczeniu sprężyny w scenie, zaś a w powyższym równaniu jest stałą liczoną według wzoru $a = \frac{\sqrt{l^2 - h^2}}{r}$, która zapewni, że sprężyna będzie miała długość l.

Wektory normalny i binormalny definiują płaszczyznę prostopadłą do krzywej. W tej płaszczyźnie na okręgu o ustalonym promieniu (grubość sprężyny ***rsmall***) będziemy ostatecznie umieszczać wierzchołki siatki, przy czym współrzędna x oryginalnej pozycji wierzchołka posłuży nam, po przemnożeniu przez 2π , za wartość kąta obrotu na tym okręgu.



1. Dodaj do projektu siatkę kwadratu, który w shaderze wierzchołków zostanie zwinięty w sprężynę: prawy przycisk na **Textured Phong, Add Model, Model**; prawy przycisk na **myModel2, Edit**, wybierz plik **tesselated_plane.3ds** (jest dołączony do materiałów do zajęć umieszczonych na stronie www); prawy przycisk na **myModel2, Rename, Plane**.
2. Dodaj nowe mapowanie strumienia wierzchołków dla tego modelu: prawy przycisk na **Textured Phong, Add Stream Mapping**; prawy przycisk na **Stream Mapping_1**, **Rename, Stream Mapping Plane**; dwuklik na **Stream Mapping Plane**, upewnij się, że na liście jest tylko jeden wpis – **POSITION**
3. Dodaj do projektu parametry, które posłużą do opisu sprężyny ***l***, ***h0***, ***r*** i ***rsmall***: prawy klawisz na **Textured Phong, Add Variable, Float, Float**; prawy przycisk na **f1MyFloat, Rename, *l***; prawy przycisk na ***l***, wpisz wartość **1000**, **OK**; powtórz to samo dla ***h0***, ***r*** i ***rsmall*** podając wartości:
   h0 = 100.0
   r = 30.0
   rsmall = 6.0
4. Uporządkujemy teraz przebiegi renderowania i dodamy jeden nowy: prawy przycisk na **Pass 0, Delete** (ostrożnie, żeby nie usunąć innego przebiegu – usuwamy ten przebieg, który był cały czas nieaktywny); prawy przycisk na **Pass 1, Rename, Pass Cube**; prawy przycisk na **Pass 2, Rename, Pass Teapot**; prawy przycisk na **Pass Teapot, Copy**; prawy przycisk na **Pass Teapot, Paste**; prawy przycisk na **Pass Teapot_1, Rename, Pass Spring**
5. Usuń nie wykorzystywane już parametry: prawy przycisk na **fSpecularPower, Delete**; tak samo **fvLightPosition** i **fvEyePosition**
6. Zmień model i mapowanie strumienia wierzchołków dla przebiegu Pass Spring: prawy prawy przycisk na **Model** w gałęzi **Pass Spring, Reference Node, Plane**; prawy przycisk na **Stream Mapping** w gałęzi **Pass Spring, Reference Node, Stream Mapping Plane**

Na tym etapie wszystkie ustawienia w projekcie są już przygotowane do renderowania (nieruchomej na razie) sprężyny. Pozostaje tylko wpisanie kodu odpowiednich obliczeń do shadera wierzchołków w przebiegu Pass Spring.

7. Przejdź do edycji shadera wierzchołków w przebiegu Pass Spring: dwuklik na **Vertex Shader** w gałęzi **Pass Spring**

8. Dodaj globalne zmienne powiązane z parametrami umieszczonymi w projekcie:

```
float l;
float h0;
float r;
float rsmall;
```

9. Dodaj następująca stałą (w zasięgu globalnym):

```
const float two_pi = 6.2831853071795864769252867665590;
```

10. Popraw deklarację struktury `VS_INPUT` tak, żeby odpowiadała mapowaniu strumienia wierzchołków dla przebiegu Pass Spring:

```
struct VS_INPUT
{
   float4 Position : POSITION0;
};
```

11. Dodaj funkcję obliczania parametru a w równaniu krzywej (wartość ta odpowiada pochodnej kąta obrotu punktu na krzywej wokół pionowej osi):

```
float dangle_ds(float l, float r, float h)
{
   return sqrt(l*l-h*h)/r;
}
```

12. Dodaj funkcje liczące punkt na krzywej, oraz pierwszą i drugą pochodną (wektor styczny i normalny):

```
float3 get_position(float s, float r, float h, float a)
{
   return float3(r*cos(s*a),-h0+s*h,r*sin(s*a));
}
float3 get_tangent(float s, float r, float h, float a)
{
   return float3(-a*r*sin(a*s), h, a*r*cos(a*s));
}
float3 get_normal(float s, float r, float h, float a)
{
   return float3(-a*a*r*cos(a*s), 0, -a*a*r*sin(a*s));
}
```

13. W funkcji `vs_main` dodaj na samym początku liczenie (na podstawie współrzędnej y wierzchołka siatki) punktu na krzywej i trzech wektorów tworzących ortogonalny układ współrzędnych styczny do niej:

```
float h = h0;   // okaże się wygodne później
float a = dangle_ds(l, r, h);
float3 CurvePosition = get_position(Input.Position.y, r, h, a);
float3 CurveTangent  = normalize(get_tangent(Input.Position.y, r, h, a));
float3 CurveNormal    = normalize(get_normal(Input.Position.y, r, h, a));
float3 CurveBinormal  = cross(CurveNormal, CurveTangent);
```

14. Policz normalną punktu w układzie sceny, obracając wektor jednostkowy w płaszczyźnie wektorów `CurveNormal` i `CurveBinormal` o kąt równy 2π razy współrzędna x wierzchołka siatki:

```
float3 Normal  =     CurveNormal * cos(two_pi * Input.Position.x) +
                   CurveBinormal * sin(two_pi * Input.Position.x);
```

15. Policz pozycję punktu na powierzchni sprężyny odsuwając się o `rsmall` wzdłuż `Normal` od `CurvePosition` (przy okazji przekształcamy float3 na float4):

```
float4 Position       = float4(CurvePosition + rsmall * Normal, 1.0);
```

16. Popraw odpowiednio obliczenia struktury `Output`:

```
Output.Position         = mul( Position, matViewProjection );
Output.Texcoord         = float2(Input.Position.x * 0.4,
Input.Position.y * 13.7);
Output.ViewDirection    = mul( float4(0,0,0,1), matViewInverse ) -
Position;
Output.Normal           = Normal;
Output.Tangent          = CurveTangent;
```

17. Przed obejrzeniem końcowego efektu przesuńmy jeszcze czajnik o 30.0 jednostek do góry zmieniając w funkcji `vs_main` w shaderze wierzchołków w przebiegu Pass Teapot linijkę:

```
Output.Position = mul( Input.Position, matViewProjection );
```

na

```
Output.Position = mul( Input.Position + float4(.0,30.0,.0,.0),
matViewProjection );
```

Po skompilowaniu shaderów i lekkim oddaleniu kamery efekt na ekranie powinien być mniej więcej taki:

Animację sprężyny uzyskamy zmieniając parametr ***h*** w równaniu krzywej na podstawie analitycznego rozwiązania równania ruchu harmonicznego z tłumieniem.
Ruch harmoniczny z tłumieniem jest rozwiązaniem równania różniczkowego zwyczajnego:

$$m\ddot{x} + d\dot{x} + kx = 0$$

gdzie x to wychylenie punktu z położenia równowagi, $\dot{x}$ prędkość, $\ddot{x}$ przyspieszenie, m masa, d współczynnik tłumienia lepkiego, k stała sprężystości. W zależności od doboru stałych mamy przypadek, gdy punkt po wychyleniu z położenia równowagi wraca do położenia równowagi i zatrzymuje się w nim (overdamping) lub gdy punkt wraca do położenia równowagi i przekracza je wielokrotnie, wykonując drgania wokół niego (underdamping). Wykorzystamy rozwiązanie analityczne dla tego drugiego przypadku postaci przy założeniu, że w chwili początkowej $t = 0$, $x(0) = 0$ oraz $\dot{x}(0) = v\max$:

$$h = h_0 + x\max * e^{-\frac{1.38629}{2} * \frac{t}{thalf}} * \sin(v\max / x\max * t)$$

W powyższym wzorze: h_0 to wartość h w położeniu równowagi, t to czas w sekundach, $thalf$ to czas malenia amplitudy drgań o połowę, $v\max$ to prędkość początkowa, $x\max$ to maksymalne wychylenie, zaś $1.38629 \approx -2\ln(0.5)$.

18. Dodaj do projektu parametr *time*, który będzie odliczał czas w sekundach. Ustal cykl na 10 sekund (po tym okresie *time* będzie zerowane).: prawy przycisk na **Textured Phong, Add Variable, Float, Predefined, fTime0_X**; prawy przycisk na **fTime0_X, Rename, *time***; z menu programu **Edit\Preferences.**.., zakładka **General**, pole **Cycle time for pre-defined 'time' variable (sec)**, wpisz **10.0**

19. Dodaj do projektu jeszcze trzy parametry (***thalf***, ***xmax***, ***vmax***) charakteryzujące ruch harmoniczny: prawy przycisk na **Textured Phong, Add Variable, Float, Float**; prawy przycisk na **f1MyFloat, Rename, *thalf***; dwuklik na ***thalf***, wpisz wartość **3.0**, kliknij **OK**; powtórz to samo dla ***xmax*** i ***vmax*** nadając im wartości ***xmax*** = **20.0**, ***vmax*** = **150.0**

20. Dodaj do shaderów wierzchołków w przebiegach **Pass Teapot** i **Pass Spring** następujące zmienne globalne powiązane z parametrami w projekcie:

```
float time;
float xmax;
float vmax;
float thalf;
```

21. Dodaj do shaderów wierzchołków w przebiegach **Pass Teapot** i **Pass Spring** następującą funkcję:

```
float spring_solution(float t)
{
   return xmax * exp(-0.5*(1.38629/thalf)*t)*sin(vmax/xmax*t);
}
```

22. W funkcji `vs_main` w shaderze wierzchołków w przebiegu **Pass Spring** zamień:

```
  float h = h0;
```

na

```
  float h = h0+spring_solution(time);
```

23. W funkcji `vs_main` w shaderze wierzchołków w przebiegu **Pass Teapot** zamień:

```
Output.Position = mul( Input.Position + float4(.0,30.0,.0,.0),
   matViewProjection );
```

na

```
Output.Position = mul( Input.Position +
   float4(.0,30.0+spring_solution(time),.0,.0), matViewProjection );
```

24. Skompiluj i obejrzyj gotową animację.