import { useEffect, useState } from 'react';

interface GeometricShape {

id: number;

type: 'cube' | 'triangle' | 'sphere' | 'wireframe';

x: number;

y: number;

size: number;

delay: number;

rotation: number;

}

const GeometricBackground = () => {

const [shapes, setShapes] = useState<GeometricShape[]>([]);

useEffect(() => {

const generateShapes = () => {

const newShapes: GeometricShape[] = [];

for (let i = 0; i < 15; i++) {

const shapeTypes = ['cube', 'triangle', 'sphere', 'wireframe'] as const;

newShapes.push({

id: i,

type: shapeTypes[Math.floor(Math.random() \* shapeTypes.length)],

x: Math.random() \* 100,

y: Math.random() \* 100,

size: Math.random() \* 50 + 25,

delay: Math.random() \* 8,

rotation: Math.random() \* 360

});

}

setShapes(newShapes);

};

generateShapes();

}, []);

const renderShape = (shape: GeometricShape) => {

const baseClasses = "absolute opacity-15 animate-float";

const style = {

left: `${shape.x}%`,

top: `${shape.y}%`,

animationDelay: `${shape.delay}s`,

transform: `rotate(${shape.rotation}deg)`

};

switch (shape.type) {

case 'cube':

return (

<div

className={`${baseClasses} border-2 border-primary/30`}

style={{

...style,

width: `${shape.size}px`,

height: `${shape.size}px`,

background: 'linear-gradient(135deg, hsl(var(--primary) / 0.1), hsl(var(--secondary) / 0.1))',

borderRadius: '4px'

}}

/>

);

case 'sphere':

return (

<div

className={`${baseClasses} border-2 border-secondary/30 rounded-full`}

style={{

...style,

width: `${shape.size}px`,

height: `${shape.size}px`,

background: 'radial-gradient(circle, hsl(var(--secondary) / 0.2), transparent 70%)'

}}

/>

);

case 'wireframe':

return (

<div

className={`${baseClasses}`}

style={{

...style,

width: `${shape.size}px`,

height: `${shape.size}px`,

border: '2px solid hsl(var(--accent) / 0.3)',

borderStyle: 'dashed',

borderRadius: '8px',

background: 'transparent'

}}

/>

);

case 'triangle':

default:

return (

<div

className={`${baseClasses}`}

style={{

...style,

width: 0,

height: 0,

borderLeft: `${shape.size/2}px solid transparent`,

borderRight: `${shape.size/2}px solid transparent`,

borderBottom: `${shape.size}px solid hsl(var(--accent) / 0.2)`

}}

/>

);

}

};

return (

<div className="fixed inset-0 pointer-events-none overflow-hidden z-0">

{shapes.map((shape) => (

<div key={shape.id}>

{renderShape(shape)}

</div>

))}

</div>

);

};

export default GeometricBackground;