```
import * as THREE from 'three'
import * as React from 'react'
import { useLayoutEffect, useMemo, useRef } from 'react'
import { extend, ReactThreeFiber, useThree, useFrame, ThreeElements } from '@react-
three/fiber'
import { MeshBVHUniformStruct, MeshBVH, SAH } from 'three-mesh-bvh'
import { MeshRefractionMaterial as MeshRefractionMaterial_ } from
'../materials/MeshRefractionMaterial'

declare module '@react-three/fiber' {
  interface ThreeElements {
    meshRefractionMaterial: typeof MeshRefractionMaterial_
  }
}

export type MeshRefractionMaterialProps = ThreeElements['shaderMaterial'] & {
  /** Environment map */
  envMap: THREE.CubeTexture | THREE.Texture
  /** Number of ray-cast bounces, it can be expensive to have too many, 2 */
  bounces?: number
  /** Refraction index, 2.4 */
  ior?: number
  /** Fresnel (strip light), 0 */
  fresnel?: number
  /** RGB shift intensity, can be expensive, 0 */
  aberrationStrength?: number
  /** Color, white */
  color?: ReactThreeFiber.Color
  /** If this is on it uses fewer ray casts for the RGB shift sacrificing physical accuracy,
true */
  fastChroma?: boolean
}

const isCubeTexture = (def: THREE.CubeTexture | THREE.Texture): def is THREE.CubeTexture =>
  def && (def as THREE.CubeTexture).isCubeTexture

export function MeshRefractionMaterial({
  aberrationStrength = 0,
  fastChroma = true,
  envMap,
  ...props
}: MeshRefractionMaterialProps) {
  extend({ MeshRefractionMaterial: MeshRefractionMaterial_ })

  const material = useRef(null)
  const { size } = useThree()

  const defines = useMemo(() => {
    const temp = {} as { [key: string]: string }
    // Sampler2D and SamplerCube need different defines
    const isCubeMap = isCubeTexture(envMap)
    const w = (isCubeMap ? envMap.image[0]?.width : envMap.image.width) ?? 1024
    const cubeSize = w / 4
    const _lodMax = Math.floor(Math.log2(cubeSize))
    const _cubeSize = Math.pow(2, _lodMax)
    const width = 3 * Math.max(_cubeSize, 16 * 7)
    const height = 4 * _cubeSize
    if (isCubeMap) temp.ENVMAP_TYPE_CUBEM = ''
    temp.CUBEUV_TEXEL_WIDTH = `${1.0 / width}`
    temp.CUBEUV_TEXEL_HEIGHT = `${1.0 / height}`
    temp.CUBEUV_MAX_MIP = `${_lodMax}.0`
    // Add defines from chromatic aberration
    if (aberrationStrength > 0) temp.CHROMATIC_ABERRATIONS = ''
    if (fastChroma) temp.FAST_CHROMA = ''
    return temp
  }, [aberrationStrength, fastChroma])

  useLayoutEffect(() => {
    // Get the geometry of this materials parent
    const geometry = (material.current as any)?.__r3f?.parent?.object?.geometry
    // Update the BVH
    if (geometry) {
      ;(material.current as any).bvh = new MeshBVHUniformStruct()
      ;(material.current as any).bvh.updateFrom(new MeshBVH(geometry.clone().toNonIndexed(),
{ strategy: SAH }))
    }
  }, [])

  useFrame(({ camera }) => {
    ;(material.current as any)!.viewMatrixInverse = camera.matrixWorld
    ;(material.current as any)!.projectionMatrixInverse = camera.projectionMatrixInverse
  })

  return (
    <meshRefractionMaterial
      // @ts-ignore
      key={JSON.stringify(defines)}
      // @ts-ignore
      defines={defines}
      ref={material}
      resolution={[size.width, size.height]}
      aberrationStrength={aberrationStrength}
      envMap={envMap}
      {...props}
    />
  )
}
```